C programming

Lecture 4: Logical expressions, conditional statements

Based on slides designed by Shay Artsi, Aythan Avior, Gitit Rockstein and Saher Ismir for the department of Computer Science, Technion, Israel Institute of Technology
Translated and updated by Anne Weill-Zrahia
Statements

- Expression statement – terminated by `;`; usually performs some action
- Empty statement – terminated by `;`; does nothing
- Compound statement (or block) – a series of statements between `{}`
- Contents of a block may be blocks
- A block can be empty (same signification as empty statement)
Logical expressions

• Calculations are only a small part of computer programming
• Decision statements are needed - they will specify the order in which statements are executed

• A logical expression can have two values
  ■ TRUE
  ■ FALSE
  or in C language
  ■ 0 for FALSE
  ■ 1 or any number≠0 for TRUE

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

<table>
<thead>
<tr>
<th>Boolean Value</th>
<th>C Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>a</td>
</tr>
<tr>
<td>false</td>
<td>a + b</td>
</tr>
<tr>
<td>false</td>
<td>a += b</td>
</tr>
<tr>
<td>true</td>
<td>a = 2*c</td>
</tr>
</tbody>
</table>

Relational and Equality operators

```
int a = 0, b = 5;
```

### Expression | Value (int) | Value (logical)
---|---|---
```
a == b
b == a
b <= 8
a != b
0 < -b < 8
a = b
b = a
a = a
```

- **Relational**
  - `<  <=  >=  >`

- **Equality**
  - `==  !=`

- **Values:**
  - `TRUE` 1
  - `FALSE` 0
Logical operations

- Are used to build logical expressions.
- Example:
  \[(0 < b) && (b < 8)\]

<table>
<thead>
<tr>
<th>оператор</th>
<th>(logical operator)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>not</td>
</tr>
</tbody>
</table>

| \(exp_1\) | \(exp_2\) | \(exp_1 \&\& exp_2\) | \(exp_1 || exp_2\) | \(! exp_1\) |
|------------|------------|-------------------|-------------------|-------------|
| ≠0         | ≠0         | 1                 | 1                 | 0           |
| ≠0         | 0          | 0                 | 1                 | 0           |
| 0          | ≠0         | 0                 | 1                 | 1           |
| 0          | 0          | 0                 | 0                 | 1           |
## Operators table

<table>
<thead>
<tr>
<th>Group</th>
<th>Operators</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>! + - (type) sizeof() ++ --</td>
<td>Right to left</td>
</tr>
<tr>
<td>Multiplication</td>
<td>* / %</td>
<td>Left to right</td>
</tr>
<tr>
<td>Addition</td>
<td>+ -</td>
<td>Left to right</td>
</tr>
<tr>
<td>Relational</td>
<td>&lt; &lt;= &gt;= &gt;</td>
<td>Left to right</td>
</tr>
<tr>
<td>Equality</td>
<td>== !=</td>
<td>Left to right</td>
</tr>
<tr>
<td>Logical AND</td>
<td>&amp;&amp;</td>
<td>Left to right</td>
</tr>
<tr>
<td>Logical OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>= += -= *= /= %=</td>
<td>Right to left</td>
</tr>
</tbody>
</table>
Short Circuit

• Computation of logical expressions stops the moment the value is known:
• Example 1:
  \[(a \leq 3) \&\& (i++ < 100)\]
  - If \(a > 3\) obviously the whole expression is F
  - The second parenthesis will not be computed
  - The value of \(i\) will not be changed.

• Example 2:
  \[(a \leq 3) \| (i++ < 100)\]
  - If \(a \leq 3\) obviously the whole expression is T
  - The second parenthesis will not be computed
  - The value of \(i\) will not be changed
If statement

- **if** (expression) statement
  - *If the logical value of* expression *is TRUE, statement will be executed*
  - *If the logical value of* expression *is FALSE, statement will not be executed, and command will go to next statement*

- A statement can be complex:

```c
if ( b * b < 4 * a * c ) {
    printf("No real solution to the equation.\n");
    solutions = 0;
}
```
**if-else statement**

```c
if ( exp ) stat1 else stat2
```

- If the value of `exp` is **TRUE**: `stat1` will be executed.
- If the value of `exp` is **FALSE**, `stat2` will be executed.

```c
d = b * b - 4 * a * c;
if   ( d < 0 ) {
    printf("No real solution to the equation.\n");
    solutions = 0;
} else {
    printf("Exists a real solution to the equation.\n");
    solutions = 1;
}
```
Nested statements if-else

d = b * b - 4 * a * c;

if ( d < 0 )
    if ( b == 0 )
        printf("Two imaginary solutions.\n");
    else
        printf("Two complex solutions.\n");
else
    if ( d == 0 )
        printf("One real solution.\n");
    else
        printf("Two real solutions.\n");
if ( d >= 0 ) /* Need only real solutions */
    if ( d > 0 )
        printf("Solutions: %g and %g:\n", 
        (-b + sqrt(d)) / ( 2*a),
        (-b - sqrt(d)) / ( 2*a));
    else
        printf("Solution: %g\n", -b / (2*a));
Oscillating else – how to prevent errors

- In C, an oscillating if statement refers to the closest else.
- In this case the execution was correct (by sheer luck).
- This is not recommended since:
  - We may not remember it is so.
  - We may write in other programming languages where it is not so.
  - Another person reading the code may not understand it correctly.

```c
if ( d >= 0 )
    if ( d > 0 )
        printf( ... );
    else
        printf( ... );
else
    printf( ... );

if ( d >= 0 )
    if ( d > 0 )
        printf( ... );
    else
        printf( ... );
else
    /* or {} */

if ( d >= 0 )
    if ( d > 0 )
        printf( ... );
    else
        printf( ... );
}
```
What if we need the opposite ...

```c
if ( d != 0 ) /* Need only real solutions */
    if ( d > 0 )
        printf("Solutions: %g and %g:\n", \\
             (-b + sqrt(d)) / ( 2*a), \\
             (-b - sqrt(d)) / ( 2*a));
    else
        printf("Solution: %g\n", -b / (2*a));
```

енный неверно!
Oscillating else – opposite case

- As said before luck is not on our side
- How to fix that:
  - The print will occur only if \( d == 0 \).

```c
if ( d != 0 )
    if ( d > 0 )
        printf( ... );
    else /* d == 0 is false */
        printf( ... );
else
    printf( ... );
```

```c
if ( d != 0 )
    if ( d > 0 )
        printf( ... );
    else /* or {} */
        printf( ... );
else
    printf( ... );
```

```c
if ( d != 0 ) {
    if ( d > 0 )
        printf( ... );
} else
    printf( ... );
```
Series of if-else-if

```c
if ( d == 0 )
    printf("One real solution.\n");
else
    if ( d > 0 )
        printf("Two real solutions.\n");
    else
        if ( b == 0 )
            printf("Two imaginary solutions.\n");
        else
            printf("Two complex solutions.\n");
```

This one is better!
Conditional statement

if ( m > n )
    max = m;
else
    max = n;

max = m > n ? m : n;

if ( m > n )
    printf("Max is %d\n", m);
else
    printf("Max is %d\n", n);

printf("Max is %d\n", m > n ? m : n);

Ternary operation

This is an expression, not a statement.
Priority of ? Is very low, just before assignment

Associativity is right to left