CMSC 104 - Lecture 8 John Y. Park, adapted by C Grasso

Relational & Logical Operators

Relational and Logical Operators

Topics

- Relational Operators and Expressions
- The if Statement
- The if-else Statement
- Nesting of if-else Statements
- Logical Operators and Expressions
- Truth Tables

Relational Operators

- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to
- == is equal to
- != is not equal to

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

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

Expression - true or false?

- a < c
- b <= c
- c <=a
- a > b
- b >= c
- a + b >= c
- a + b == c
- a != b
- a + b != c

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

int	a = 1,	b = 2,	c = 3 ;
float	x = 3.3	3 y = 6.66	;
Express	ion Nu	<u>imeric Value</u>	<u>True/False</u>
a + b b - 2	* >		
c - b	- a		
c - a			
у – х			
y - 2	* x		

Review: Structured Programming

- All programs can be written in terms of only three control structures
 - 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

```
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 104 C Coding Standards.



```
if ( age >= 18 ) {
    printf(" Go Vote!\n") ;
}
if ( value == 0 ) {
    printf ("The value you entered was zero.\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

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

Note that there is no condition for the else.



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

Another Example

if (value == 0) {

printf ("The value you entered was zero.");
} else {
 printf ("Value = %d. \n", value);
}

Nesting of if-else Statements

if (condition,) else if (condition,) /* more else-if clauses may be here */ else /* the **default case** */ . . . }

Nesting of if-else Statements

if (x == 1)
{
 ...
}
else if (x == 2)
{
 ...
}
else
{
 ...
}

if (x == 1)else /* x != 1 */ if (y == 'b')

Example

```
if ( value == 0 )
{
    printf ("The value you entered was 0") ;
}
else if ( value < 0 )
{
    printf ("%d is negative.\n", value) ;
}
else
{
    printf ("%d is positive.\n", value) ;
}</pre>
```

Gotcha! = versus ==

```
int a = 2;
```

```
if ( a = 1 ) { // semantic (logic) error!
    printf ("a is one \n") ;
}
else if ( a == 2 )
{
    printf ("a is two\n") ;
}
else
{
    printf ("a is %d\n", a) ;
}
```

Gotcha (con't)

- The statement if (a = 1) is syntactically correct,
 - No error message will be produced.
 - Some compilers will produce a warning.
 - However, a semantic (logic) error mayoccur.
- An assignment expression has a value -- the value being assigned.
 - In this case the value being assigned is 1, which is true.

Gotcha (con't)

- 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 &&

if (age < 1 && gender == `f')
{
 printf ("You have a baby girl! \n");
}</pre>

Truth Table for &&

<u>Exp₁</u>	<u>Exp</u> 2	<u>Exp₁ && Exp₂</u>
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

 $Exp_1 \&\& Exp_2 \&\& \dots \&\& Exp_n$ will evaluate to 1 (true) only if ALL **subconditions** are true.

Example Use of ||

```
if (grade == `D' || grade == `F')
{
    printf ("See you next semester! \n");
}
```

Truth Table for ||

<u>Exp₁</u>	<u>Exp</u> 2	<u>Exp₁ && Exp₂</u>
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

 $Exp_1 \&\& Exp_2 \&\& \dots \&\& Exp_n$ will evaluate to 1 (true) if only ONE subcondition is true.

Example Use of !

```
if ( ! (x == 2) ) // same as (x != 2)
{
    printf("x is not equal to 2.\n");
}
else
{
    printf("x is equal to 2.\n");
}
```

Truth Table for !

<u>Exp₁</u>	<u>! Exp₁</u>	
0	1	
nonzero	0	

Operator Precedence and Associativity

Precedence

```
()
!(not) - (negation)
* / %
+ (addition) - (subtraction)
< <= > >=
== !=
&&
||
=
```

<u>Associativity</u>

left to right/inside-out right to left left to right right to left

Some Practice Expressions

int a = 1, b = 0, c = 7;

<u>Expression</u>	<u>Numeric Value</u>	<u>True/False</u>
a		
b		
C		
a + b		
a && b		
a b		

Some Practice Expressions

int a = 1, b = 0, c = 7;

Expression Numeric Value True/False !c !!c a && !b a < b && b < c a > b && b < c a >= b || b > c

More Practice

Given int a = 5, b = 7, c = 17;

evaluate each expression as True or False.

$$c / b == 2$$

 $c % b <= a % b$
 $b + c / a != c - a$
 $(b < c) & (c == 7)$
 $(c + 1 - b == 0) || (b = 5)$