

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

Reading

- Sections 2.6, 4.10, 4.11



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	Value	Expression	Value
a < c		a + b >= c	
b <= c		a + b == c	
c <= a		a != b	
a > b		a + b != c	
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 = 3.33, y = 6.66 ;
```

<u>Expression</u>	<u>Numeric Value</u>	<u>True/False</u>
a + b		
b - 2 * a		
c - b - a		
c - a		
y - x		
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.

Examples



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

```
if ( value == 0 )
{
    printf ("The value you entered was 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 5 spaces -- be consistent!



Selection: the **if-else** statement

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



Example

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



Example

```
if ( value == 0 )
{
    printf ("The value you entered was zero.\n");
    printf("Please try again.\n");
}
else
{
    printf ("Value = %d.\n", value);
}
```



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

```
if ( condition1 )
{
    statement(s)
}
else if ( condition2 )
{
    statement(s)
}
... /* more else clauses may be here */
else
{
    statement(s) /* the default case */
}
```



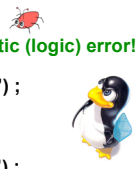
Example

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



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, 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**.

<code>&&</code>	is AND if (<code>x > 5 && y < 6</code>)
<code> </code>	is OR if (<code>z == 0 x > 10</code>)
<code>!</code>	is NOT if (<code>!(bob > 42)</code>)
- 

Example Use of &&

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



Truth Table for &&

Exp ₁	Exp ₂	Exp ₁ && Exp ₂
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1



Exp₁ && Exp₂ && ... && 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 ||

Exp ₁	Exp ₂	Exp ₁ Exp ₂
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1



Exp₁ || Exp₂ || ... || 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") ;  
}
```

Truth Table for !

Expression	! Expression
0	1
nonzero	0



Operator Precedence and Associativity

Precedence

()
 ++ -- ! + (unary) - (unary) (type)
 * / %
 + (addition) - (subtraction)
 < <= > >=
 == !=
 &&
 ||
 = += -= *= /= %=
 , (comma)

Associativity

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



Some Practice Expressions

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

Expression	Numeric Value	True/False
a		
b		
c		
a + b		
a && b		
a b		
!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)