

The Nature of Truth

- In C++, zero is considered false
- and all other values are considered true, although true is usually represented by 1
- If a statement is true, all you know is that it is nonzero, and any nonzero statement is true.

Relational Operators

- The relational operators are used to determine whether two numbers
 - are equal,
 - or if one is greater or less than the other.
 - every relational statement evaluates to either 1 (TRUE) or 0 (FALSE)
 - `myAge == yourAge`; // is the value in myAge the same as in yourAge?
 - `myAge > yourAge`; // is myAge greater than yourAge?

The Relational Operators

| • <i>Name</i> | <i>Operator</i> | <i>SampleEvaluates</i> |
|---------------------------------|--------------------|--|
| • <i>Equals</i> | <code>==</code> | <code>100 == 50;false</code> <code>50 == 50;true</code> |
| • <i>Not Equals</i> | <code>!=</code> | <code>100 != 50;true</code> <code>50 != 50;false</code> |
| • <i>Greater Than</i> | <code>></code> | <code>100 > 50;true</code> <code>50 > 50;false</code> |
| • <i>Greater Than or Equals</i> | <code>>=</code> | <code>100 >= 50;true</code> <code>50 >= 50;true</code> |
| • <i>Less Than</i> | <code><</code> | <code>100 < 50;false</code> <code>50 < 50;false</code> |
| • <i>Less Than or Equals</i> | <code><=</code> | <code>100 <= 50;false</code> <code>50 <= 50;true</code> |

The if Statement

- test for a condition (such as whether two variables are equal) and
- branch to different parts of your code, depending on the result.
- if (expression)
statement;
- If the expression has the value 0, it is considered false, and the statement is skipped
- If the expression has any nonzero value, it is considered true, and the statement is executed.

if Statement: Examples

- `if (bigNumber > smallNumber)`
`bigNumber = smallNumber;`

This code compares `bigNumber` and `smallNumber`

If `bigNumber` is larger, the second line sets its value to the value of `smallNumber`.

if Statement: Examples

- `if (expression)`
`{ statement1;`
`statement2;`
`statement3; }`

A block of statements surrounded by braces is exactly equivalent to a single statement

- `if (bigNumber > smallNumber)`
`{ bigNumber = smallNumber;`
`cout << "bigNumber: " << bigNumber << "\n";`
`cout << "smallNumber: " << smallNumber << "\n"; }`

Indentation Styles

- Three favorite variations

```
if (expression) {  
    statements  
}
```

```
if (expression)  
{  
    statements  
}
```

```
if (expression)  
{  
    statements  
}
```

else

- program will
 - take one branch if your condition is true
 - take another branch if your condition is false

```
if (expression)  
    statement;
```

```
else  
    statement;  
    next statement;
```

The if Statement: Syntaxes

- Form 1:
 - If the expression is evaluated as TRUE, the statements within the braces are executed and the program continues with the next statement
 - If the expression is evaluated as FALSE, the statements within the braces are NOT executed and the program JUMPS to the next statement
- if (expression)
 - {
 - statement;
 - statement;
 - statement;
 - }
 - next statement;

- Form 2:
 - If the expression is evaluated as TRUE, the statement 1's within the first pair of braces are executed and the program then JUMPS to the next statement
 - If the expression is evaluated as FALSE, the statement 1's within the braces are NOT executed and the program JUMPS to statement 2's within the second pair of braces and the program then continues to the next statement
- ```
if (expression)
{ statement 1;
 statement 1;
 statement 1;
}
else
{ statement 2;
 statement 2;
 statement 2;
}
next statement;
```

## Advanced/Complex IF

- any statement can be used in an if or else clause, even another if or else statement
- if (expression1)  
    { if (expression2)  
        statement1;  
    else  
        { if (expression3)  
            statement2;  
        else  
            statement3;  
        }  
    }  
else  
statement4;

## Looping/Iteration

- doing the same thing again and again
- many programming problems are solved by repeatedly acting on the same data
- the principal method of iteration is the loop

## while Loops

- causes a program to repeat a sequence of statements as long as the starting condition remains true

```
while (condition)
statement;
```

```
while (condition)
{
 statement;
 statement;
 statement;
}
next statement;
```

## Example of while loop

```
// count to 10
int x = 0;
while (x < 10)
 cout << "X: " << x++;
```

## **continue and break**

- **continue;**
  - causes a while or for loop to begin again at the top of the loop.
- **break;**
  - causes immediate exit from the while loop, and program execution resumes after the closing brace

```
while (condition)
{
 if (condition2)
 break;
 statements;
 statements;
 statements;
}
```

## **while (1) Loops**

- You can create a loop that will never end by using the number 1 for the condition to be tested.
- Since 1 is always true, the loop will never end, unless a break statement is reached.



## Example of while true loop

// Demonstrates a while true loop

```
#include <iostream.h>
```

```
int main()
{
 int counter = 0;
 while (1)
 {
 counter ++;
 if (counter > 10)
 break;
 }
 cout << "Counter: " << counter << "\n";
 return 0;
}
```

Output: Counter: 11

## do...while Loops

- It is possible that the body of a while loop will never execute. The while statement checks its condition before executing any of its statements, and if the condition evaluates false, the entire body of the while loop is skipped
- The do...while loop executes the body of the loop before its condition is tested and ensures that the body always executes at least one time

## The do...while Statement

```
do
statement
while (condition);
next statement;
```

```
do
{
 statement;
 statement;
 statement;
}
while (condition);
next statement;
```

## Example of do while

```
// count to 10
int x = 0;
do
cout << "X: " << x++;
while (x < 10);

// print lowercase alphabets.
char ch = 'a';
do
{
 cout << ch << ' ';
 ch++;
} while (ch <= 'z');
```

## for Loops

```
for (initialization; test; action)
statement;
```

```
for (initialization; test; action)
{
statement;
statement;
statement;
}
```

## Working of for loop

- A for loop works in the following sequence:
  1. Performs the operations in the initialization.
  2. Evaluates the condition.
  3. If the condition is TRUE, executes the action statement and the loop
- After each time through, the loop repeats steps 2 and 3.

## Examples of for loop

```
// print Hello ten times
```

```
for (int i = 0; i<10; i++)
```

```
cout << "Hello! ";
```

```
for (int i = 0; i < 10; i++)
```

```
{
```

```
 cout << "Hello!" << endl;
```

```
 cout << "the value of i is: " << i << endl;
```

```
}
```

## Advanced for loops

```
// demonstrates multiple statements in for loops
```

```
#include <iostream.h>
```

```
int main()
```

```
{
```

```
 for (int i=0, j=0; i<3; i++, j++)
```

```
 cout << "i: " << i << " j: " << j << endl;
```

```
 return 0;
```

```
}
```

Output:

i: 0 j: 0

i: 1 j: 1

i: 2 j: 2

## Null statements in for loops.

```
int counter = 0;
for(; counter < 5;)
{
 counter++;
 cout << "Looping! ";
}
```

output:

Looping! Looping! Looping! Looping! Looping!

## Empty body of for Loops

```
#include <iostream.h>
```

```
int main()
{
 for (int i = 0; i<4; cout << "i: " << i++ << endl)
 ;
 return 0;
}
```

Output:

i: 0

i: 1

i: 2

i: 3

## empty for loop statement

- it is possible, using break and continue, to create a for loop with none of the three statements of for loops

```
int counter=0, max=5; // initialization
for (;;) // a for loop that doesn't end
{
 if (counter < max) //test
 {
 cout << "Hello!\n";
 counter++; // increment
 }
 else
 break;
}
```