Statements

Simple Statements

Empty statement

```
;                                   // ; is a terminator, not a separator
```

Assigment

```
x = 3 * y + 1;
```

Method call

```
string s = "a,b,c";
string[] parts = s.Split(',');      // invocation of an object method (non-static)

s = String.Join(" + ", parts);  // invocation of a class method (static)
```

if Statement


```
if ('0' <= ch && ch <= '9')
      val = ch - '0';
else if ('A' <= ch && ch <= 'Z')
      val = 10 + ch - 'A';
else {
      val = 0;
      Console.WriteLine("invalid character " + ch);
}
```

switch Statement


```
switch (country) {
      case "England": case "USA":
            language = "English";
            break;
      case "Germany": case "Austria": case "Switzerland":
            language = "German";
            break;
      case null:
            Console.WriteLine("no country specified");
            break;
      default:
            Console.WriteLine("don't know the language of " + country);
            break;
}
```

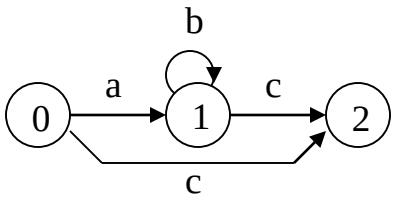
Type of the switch expression
integer type, char, enum or string (null ok as a case label).

No fall-through (unlike in C or in Java)**!**
Every statement sequence in a case must be terminated with break (or return, goto, throw).

If no case label matches ➔ default
If no default specified ➔ continuation after the switch statement

switch with Gotos

E.g. for the implementation of automata




```
int ch = Console.Read();
int state = startTab[ch];
switch (state) {
    case 0:  if (ch == 'a') { ch = Console.Read(); goto case 1; }
             else if (ch == 'c') goto case 2;
             else goto default;
    case 1:  if (ch == 'b') { ch = Console.Read(); goto case 1; }
             else if (ch == 'c') goto case 2;
             else goto default;
    case 2:  Console.WriteLine("input valid");
             break;
    default: Console.WriteLine("illegal character " + ch);
             break;
}
```

Loops

while

```
while (i < n) {
      sum += i;
      ++;
}
```

do while

```
do {
      sum += a[i];
      i--;
} while (i > 0);
```

for

```
for (int i = 0; i < n; i++)
      sum += i;
```

short form for

```
int i = 0;
while (i < n) {
    sum += i;
    i++;
}
```

foreach Statement

For iterating over collections and arrays

```
int[] a = {3, 17, 4, 8, 2, 29};
foreach (int x in a) sum += x;
```

```
string s = "Hello";
foreach (char ch in s) Console.WriteLine(ch);
```

```
Queue q = new Queue(); // elements are of type object
q.Enqueue("John"); q.Enqueue("Alice"); ...
foreach (string s in q) Console.WriteLine(s);
```

Jumps

break;	For exiting a loop or a switch statement. There is no break with a label like in Java (use *goto* instead).
continue;	Continues with the next loop iteration.
goto case 3:	Can be used in a switch statement to jump to a case label.

```
myLab:
...
goto myLab;
```

Jumps to the label *myLab*.

Restrictions:

- no jumps into a block
- no jumps out of a finally block of a try statement

return Statement

Returning from a void method

```
void Foo (int x) {
    if (x == 0) return;
    ...
}
```

Returning a value from a function method

```
int Max (int a, int b) {
    if (a > b) return a; else return b;
}
```

```
class C {
    static int Main() {
        ...
        return errorCode;       // The Main method can be declared as a function;
    }                           // the returned error code can be checked with the
                                // system variable errorlevel
}
```

Output to the Console

Examples

```
Console.Write(intVal);                  // overloaded for all primitive types
Console.WriteLine(intVal);              // for objects ToString() is called automatically

Console.Write("Hello {0}", name);       // placeholder
Console.WriteLine("{0} = {1}", x, y);
```

Placeholder syntax

"{" n ["," width] [":" format [precision]] "}"

n	argument number (starting at 0)
width	field width (exceeded if too small) positive = right-aligned, negative = left-aligned
format	formatting code (e.g. d, f, e, x, ...)
precision	number of fractional digits (sometimes number of digits)

Example: {0,10:f2}

Formatting Codes for Numbers

d, D	decimal format (integer number with leading zeroes) precision = number of digits	-xxxxx
f, F	fixed-point format precision = number of fractional digits (default = 2)	-xxxxx.xx
n, N	number format (with separator for thousands) precision = number of fractional digits (default = 2)	-xx,xxx.xx
e, E	floating-point format (case is significant) precision = number of fractional digits	-x.xxxE+xxx
c, C	currency format precision = number of fractional digits (default = 2) negative values are enclosed in brackets	$xx,xxx.xx ($xx,xxx.xx)
x, X	hexadecimal format (case is significant) precision = number of hex digits (maybe leading 0)	xxx
g, G	general (most compact format for the given value; default)	

Examples

int x = 17;

```
Console.WriteLine("{0}", x);          17
Console.WriteLine("{0,5}", x);           17

Console.WriteLine("{0:d}", x);        17
Console.WriteLine("{0,5:d3}", x);       017

Console.WriteLine("{0:f}", x);        17.00
Console.WriteLine("{0:f1}", x);       17.0

Console.WriteLine("{0:E}", x);        1.700000E+001
Console.WriteLine("{0:e1}", x);       1.7e+001

Console.WriteLine("{0:x}", x);        11
Console.WriteLine("{0:x4}", x);       0011
```

String Formatting

With *ToString* for numeric types (*int*, *long*, *short*, ...):

```
string s;
int i = 12;
s = i.ToString();              // "12"
s = i.ToString("x4");          // "000c"
s = i.ToString("f");           // "12.00"
```

With *String.Format* for arbitrary types

```
s = String.Format("{0} = {1,6:x4}", name, i);      // "val =    000c"
```

Culture-specific formatting

```
s = i.ToString("c");                                // "$12.00"
s = i.ToString("c", new CultureInfo("en-GB"));      // "£12.00"
s = i.ToString("c", new CultureInfo("de-AT"));      // "€12.00"
```

Formatted Output to a File

```
using System;
using System.IO;

class Test {

   static void Main() {
      FileStream s = new FileStream("output.txt", FileMode.Create);
      StreamWriter w = new StreamWriter(s);

      w.WriteLine("Table of sqares:");
      for (int i = 0; i < 10; i++)
         w.WriteLine("{0,3}: {1,5}", i, i*i);

      w.Close();
   }

}
```

It is not possible to have multiple *StreamWriters* working on the same stream at the same time.

Keyboard Input

int ch = Console.Read();

returns the next character.

waits until the user pressed the return key.
e.g. input: "abc" + return key.
Read returns: 'a', 'b', 'c', '\r', '\n'.

after the last character (Ctrl-Z + return) Read returns -1

string line = Console.ReadLine();

returns the next line (after Ctrl-Z+CR+LF it returns null).

waits until the user pressed the return key.
returns the line without CR, LF.

There is no *Tokenizer* for formatted input like in Java.

Input from a File


```
using System;
using System.IO;

class Test {

   static void Main() {
      FileStream s = new FileStream("input.txt", FileMode.Open);
      StreamReader r = new StreamReader(s);

      string line = r.ReadLine();
      while (line != null) {
         ...
         line = r.ReadLine();
      }

      r.Close();
   }

}
```

It is not possible to have multiple *StreamReaders* working on the same stream at the same time.

Reading Command-line Parameters


```
using System;

class Test {

   static void Main(string[] arg) {                  // e.g. invoked as: Test value = 3
      for (int i = 0; i < arg.Length; i++)
         Console.WriteLine("{0}: {1}", i, arg[i]);   // output (tokens are separated by blanks):
                                                     //  0: value
                                                     //  1: =
                                                     //  2: 3
      foreach (string s in arg)
         Console.WriteLine(s);                       // output:
                                                     //  value
                                                     //  =
                                                     //  3
   }
}
```