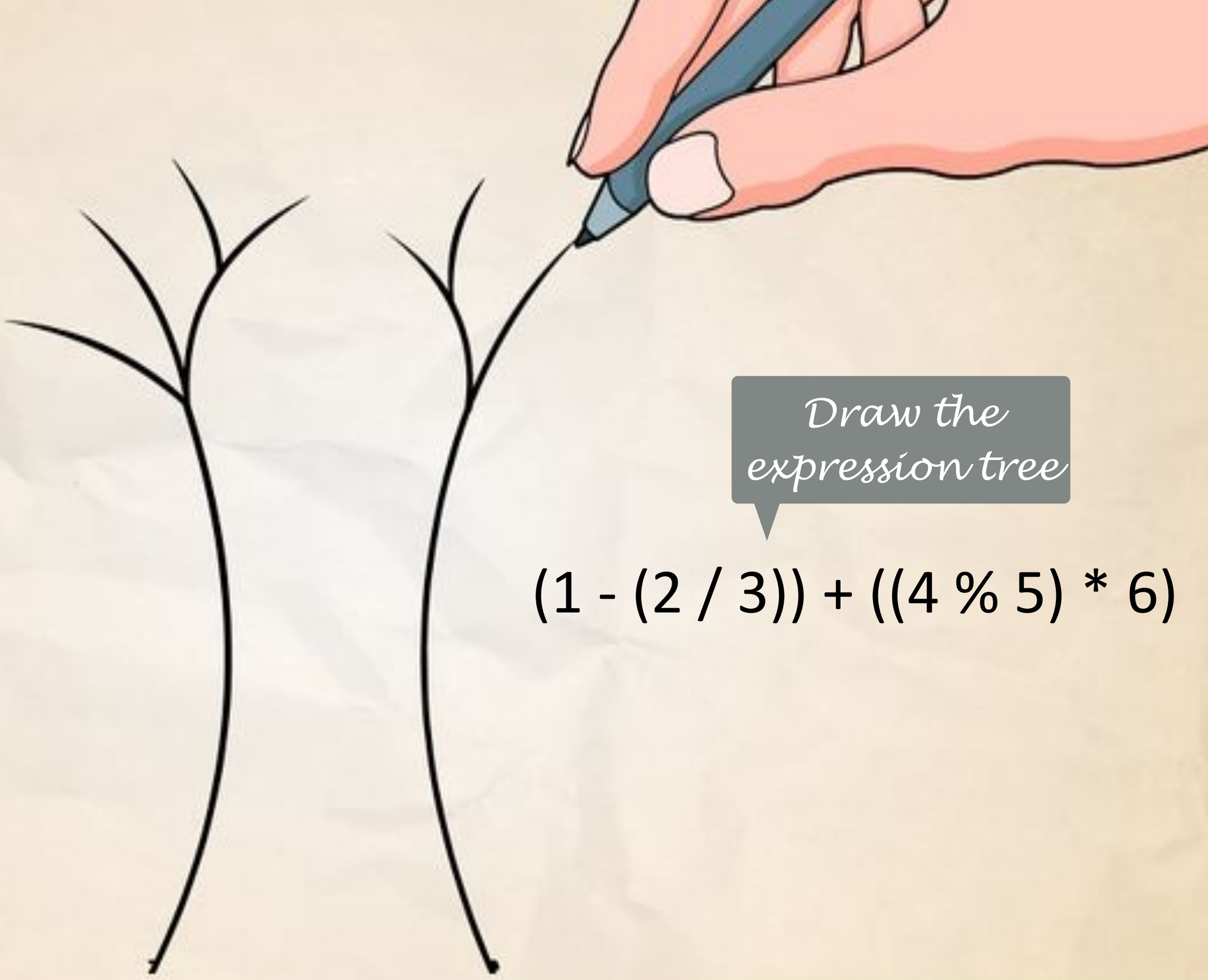
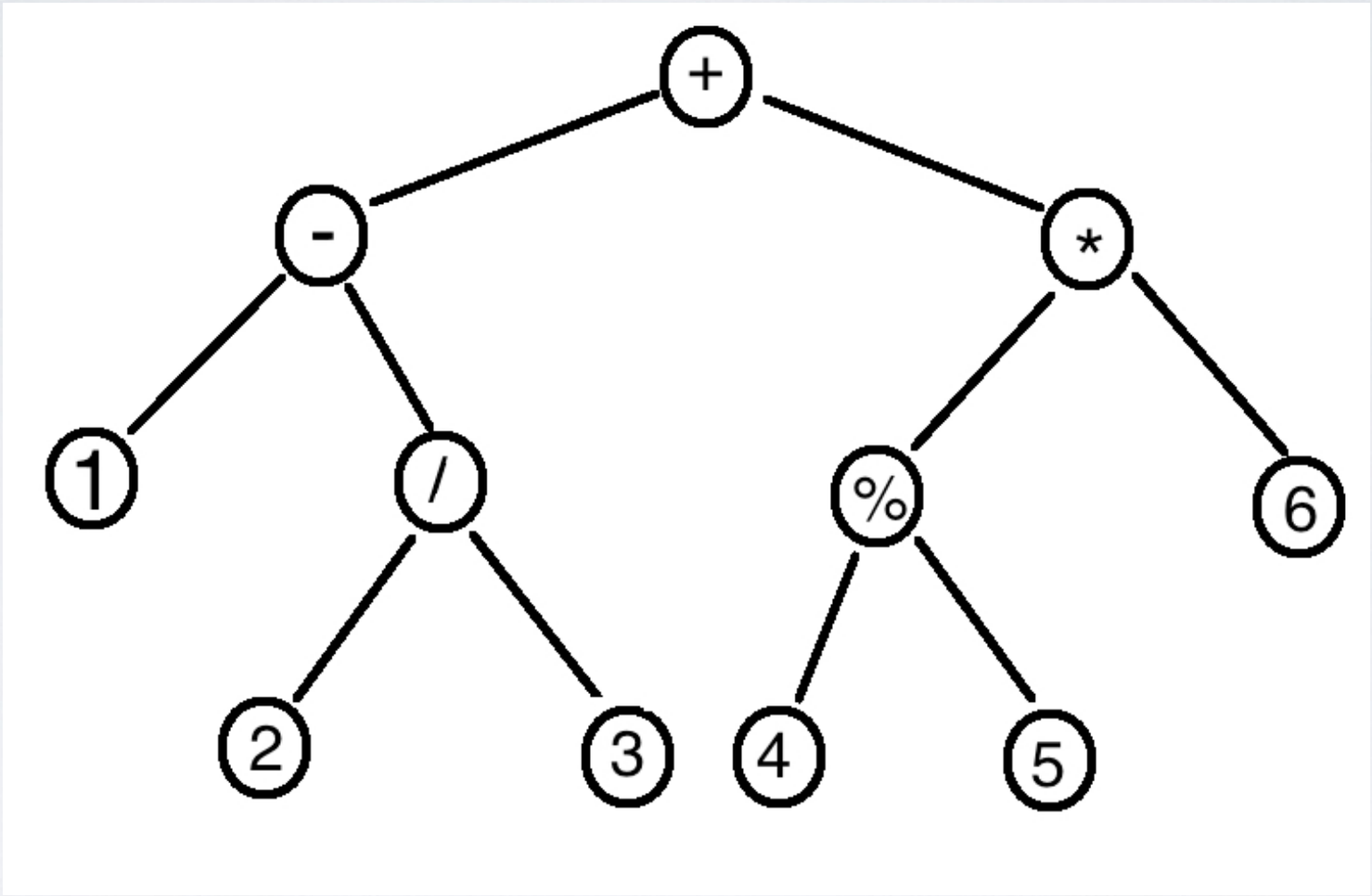
CIL BY EXAMPLE

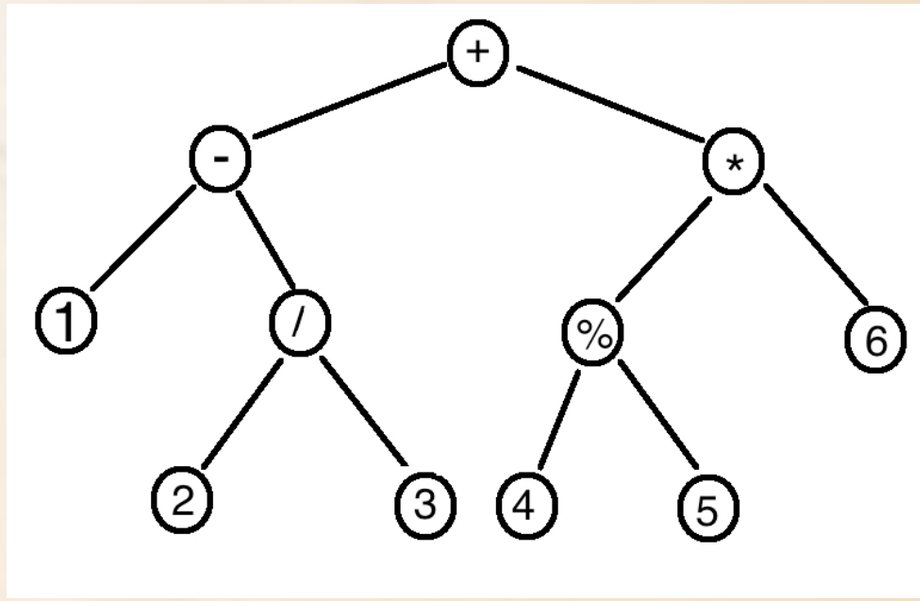
GANESH SAMARTHYAM

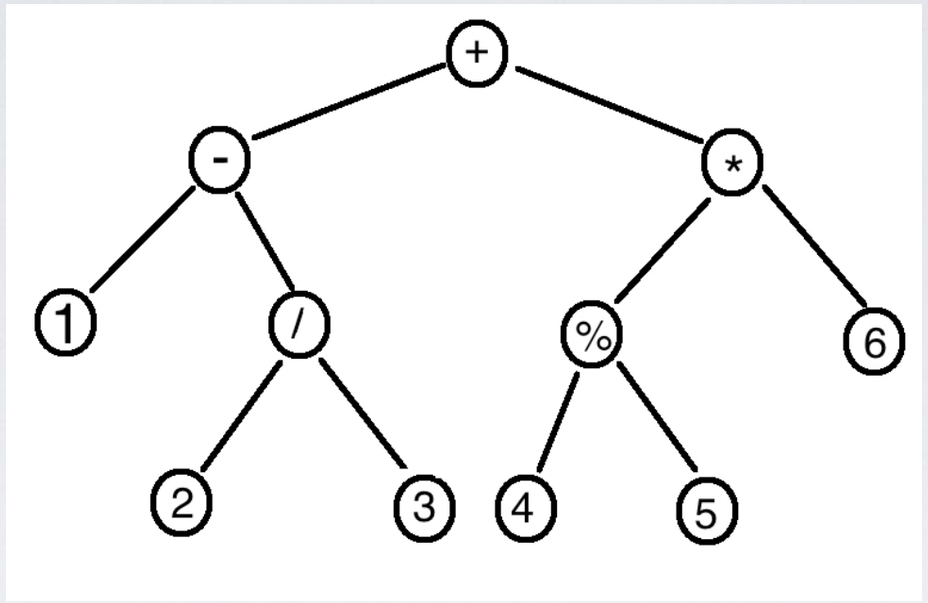
GANESH@CODEOPS.TECH


Draw the expression tree
(1 - (2 / 3)) + ((4 % 5) * 6)


+
-
*
1
/
%
6
2
3
4
5

Perform post-order traversal of the tree





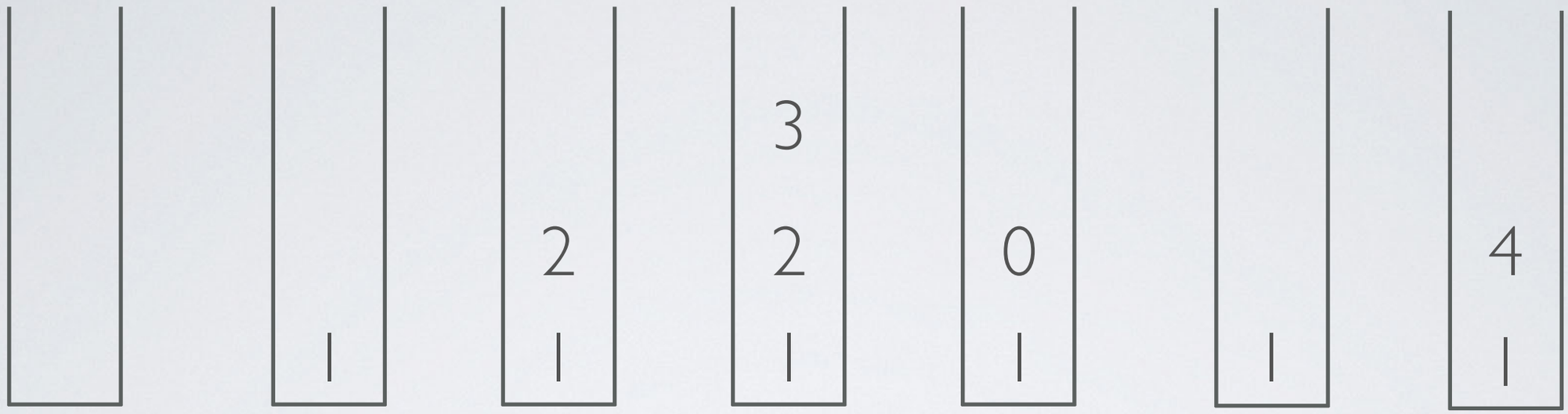
post-order
traversal
result

1 2 3 / - 4 5 % 6 * +

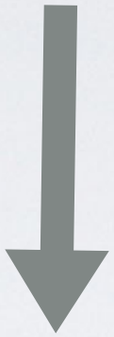
Use a stack for evaluating this postfix expression

1 2 3 / - 4 5 % 6 * +

1 2 3 / - 4 5 % 6 * +


Initial
empty
1
push 1
2
1
push 2
3
2
1
push 3
0
1
pop 3
pop 2
push 2 / 3
1
pop 0
pop 1
push 1 - 0
4
1
push 4
5
4
1
push 5
4
1
pop 5
pop 4
push 4 % 5
6
4
1
push 6
24
1
pop 6
pop 4
push 6 * 4
25
pop 24
pop 1
push 24 + 1

1 2 3 / - 4 5 % 6 * +



Let us give
names to these
operations

```
push 1
push 2
push 3
div
sub
push 4
push 5
mod
push 6
mul
add
```

```
int a = 1, b = 2, c = 3, d = 4, e = 5, f = 6;
int r = (a - (b / c)) + ((d % e) * f);
```

This is what a C# compiler generates

our bytecode

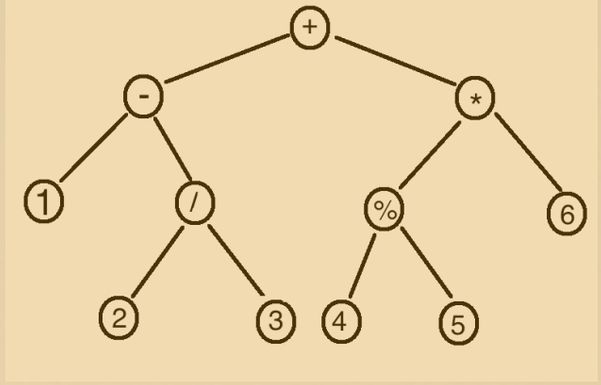
```
push 1
push 2
push 3
div
sub
push 4
push 5
mod
push 6
mul
add
```

.NET bytecode

```
ldloc.0
ldloc.1
ldloc.2
div
sub
ldloc.3
ldloc.s 4
rem
ldloc.s 5
mul
add
```

Source code

(1 - (2 / 3)) + ((4 % 5) * 6)



Compiler

CIL (Common Intermediate Language) code

```
ldloc.0
ldloc.1
ldloc.2
div
sub
ldloc.3
ldloc.s 4
rem
ldloc.s 5
mul
add
```

.NET runtime

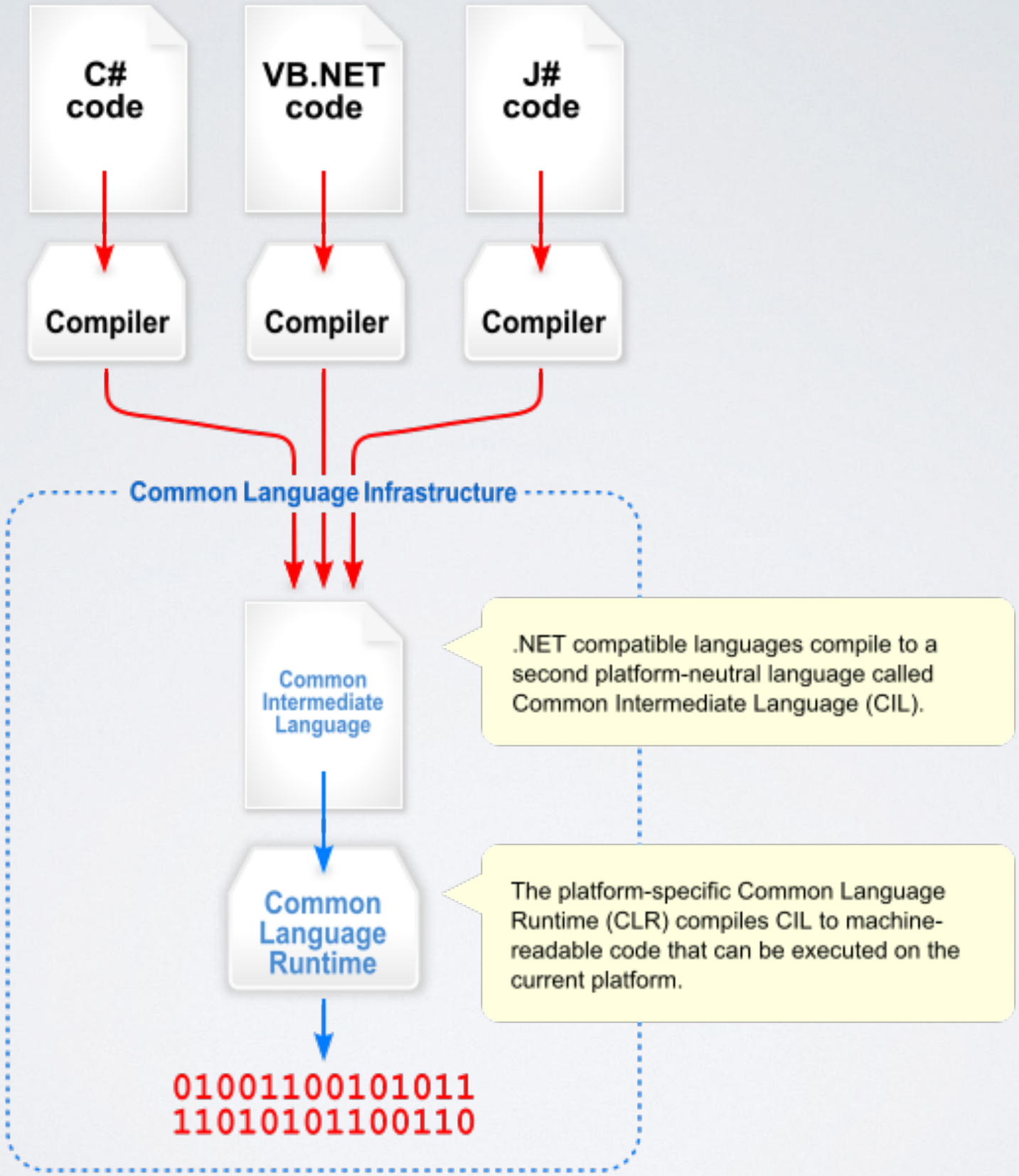
ldc.i4.s	this IL instruction refers to "load constant of type integer which is a signed one" - here it is the variable srt. This instruction leads to pushing the integer constant 10 on the stack.
stloc.0	stands for "store the stack top value to variable at location zero", here it is integer constant 10 that is popped from the stack and stored in the short variable named srt (which is indicated as the variable available in location 0).
ldloc.0	stands for "load the value of variable in location 0 to stack top". Here, the value of srt variable is pushed into the stack.
box	stands for "box the value in stack top to object type". Here the short value is converted to object type and it is now available in the top of the stack.
stlock.1	pops the object from the stack and stores it in a temporary variable that is to be returned as the return value from the method.
ldloc.1	the value in temporary variable is pushed into the stack.
ret	stands for "return the control from the method".

MSIL/CIL supports:

- Object oriented programming

- Works in terms of the data types available in the .NET Framework (e.g., System.String and System.Int32)

- Instructions can be classified into various types such as:
  - loading (ld*)
  - storing (st*)
  - method invocation
  - arithmetic operations
  - logical operations
  - control flow
  - memory allocation
  - exception handling



Source: https://en.wikipedia.org/wiki/Common_Language_Infrastructure#/media/File:Overview_of_the_Common_Language_Infrastructure.svg

Use ildasm (Windows) or monodic (Mac/Linux/…)

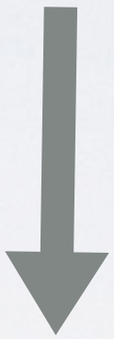
```
$ cat expr.cs
using System;
    class Hello
    {
        static void Main()
        {
           int a = 1, b = 2, c = 3, d = 4, e = 5, f = 6;
           int r = (a - (b / c)) + ((d % e) * f);
            Console.WriteLine(r);
        }
    }

$ mcs expr.cs

$ mono expr.exe
25

$ monodis --method expr.exe
Method Table (1..2)
########## .Hello
1: instance default void '.ctor' ()  (param: 1 impl_flags: cil managed )
2: default void Main ()  (param: 1 impl_flags: cil managed )

$
```

mono compiler

mono JIT/ AoT compiler

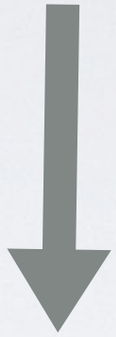
mono disassembler

```
Console.WriteLine("hello world");
```




```
// disassembled code using ildasm tool
ldstr      "hello world"
call       void [mscorlib]System.Console::WriteLine(string)
```

```
int i = 10;
if(i != 20)
        i = i*20;
Console.WriteLine(i);
```



ildasm/monodis

```
IL_0000:  ldc.i4.s   10
IL_0002:  stloc.0
IL_0003:  ldloc.0
IL_0004:  ldc.i4.s   20
IL_0006:  beq.s      IL_000d
IL_0008:  ldloc.0
IL_0009:  ldc.i4.s   20
IL_000b:  mul
IL_000c:  stloc.0
IL_000d:  ldloc.0
IL_000e:  call       void [mscorlib]System.Console::WriteLine(int32)
```

```
$ cat check.cs
using System;
class Check
{
     static void Main()
    {
          int i = 10;
          object o1 = i, o2 = i;
          if(o1 == o2)
               Console.WriteLine("yes, o1 == o2");
          else
               Console.WriteLine("no, o1 != o2!!!");
    }
}
$ mcs check.cs
$ mono check.exe
no, o1 != o2!!!
```

```
int i = 10;
object o1 = i, o2 = i;
if(o1 == o2)
	Console.WriteLine("yes, o1 == o2");
else
	Console.WriteLine("no, o1 != o2!!!");
```

ildasm/monodis

```
IL_0000:  ldc.i4.s 0x0a
IL_0002:  stloc.0
IL_0003:  ldloc.0
IL_0004:  box [mscorlib]System.Int32
IL_0009:  stloc.1
IL_000a:  ldloc.0
IL_000b:  box [mscorlib]System.Int32
IL_0010:  stloc.2
IL_0011:  ldloc.1
IL_0012:  ldloc.2
IL_0013:  bne.un IL_0027

IL_0018:  ldstr "yes, o1 == o2"
IL_001d:  call void class [mscorlib]System.Console::WriteLine(string)
IL_0022:  br IL_0031

IL_0027:  ldstr "no, o1 != o2!!!"
IL_002c:  call void class [mscorlib]System.Console::WriteLine(string)
IL_0031:  ret
```

Since boxing is done twice, the two objects o1 and o2 are allocated in two different places on the heap!

```
int i = 10;
object o1 = i, o2 = o1;
if(o1 == o2)
    Console.WriteLine("yes, o1 == o2");
else
    Console.WriteLine("no, o1 != o2!!!");
```

ildasm/monodis

```
IL_0000:  ldc.i4.s 0x0a
IL_0002:  stloc.0
IL_0003:  ldloc.0
IL_0004:  box [mscorlib]System.Int32
IL_0009:  stloc.1
IL_000a:  ldloc.1
IL_000b:  stloc.2
IL_000c:  ldloc.1
IL_000d:  ldloc.2
IL_000e:  bne.un IL_0022

IL_0013:  ldstr "yes, o1 == o2"
IL_0018:  call void class [mscorlib]System.Console::WriteLine(string)
IL_001d:  br IL_002c

IL_0022:  ldstr "no, o1 != o2!!!"
IL_0027:  call void class [mscorlib]System.Console::WriteLine(string)
IL_002c:  ret
```

Since boxing is done only once, both o1 and o2 refer to the same object; hence we get "yes, o1 == o2" printed

```
using System;
using SomeProject;

class Test
{
   public static void Main()
   {
       int i = (int) GetInfo.GetValue();
       Console.WriteLine("value of i is {0}",i);
   }
}
```

Assume that we have GetValue() method that returns an object in SomeProject.GetInfo

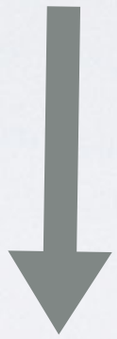
```
Unhandled Exception:
System.InvalidCastException: Specified cast is not valid.
```

How to debug when source code for SomeProject is not available?

```
// method line 4
.method public static hidebysig
           default object GetValue ()  cil managed
{
    // Method begins at RVA 0x2088
    // Code size 10 (0xa)
   .maxstack 1
   .locals init (
     int16 V_0)
    IL_0000:  ldc.i4.s 0x0a
    IL_0002:  stloc.0
    IL_0003:  ldloc.0
    IL_0004:  box [mscorlib]System.Int16
    IL_0009:  ret
} // end of method GetInfo::GetValue
```

By analysing CIL code, we find that its a short, and not a int value; hence the cast (int) fails by throwing System.InvalidCastException

```
int i = (int) GetInfo.GetValue();
```



```
int i = (int) (short) GetInfo.GetValue();
// or as
short i = (short) GetInfo.GetValue();
```

ucode

p-code

java bytecode

Other ILs: examples

dalvik bytecode

uncol

python bytecodes

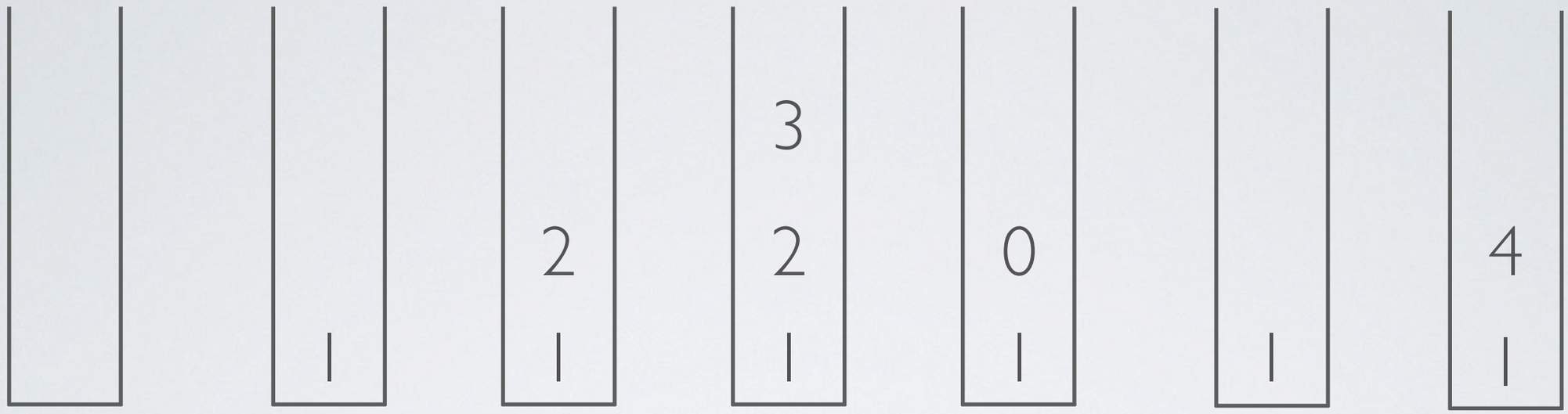
```
.method public static hidebysig
                default object GetValue ()  cil managed
```

.method	this directive refers to the fact that GetValue is a method
Public	the method's access specifier - it is publicly accessible outside the class
Hidebysig	refers to the fact that this method hides any method from the base class (if any) with the same signature
Static	the method is a static method (not an instance method)
Object	return the type of the GetValue method
GetValue()	the name and arguments (here it has no arguments) of the method
Cil managed	the code is managed code (as opposed to unmanaged code)
Maxstack	the runtime stack size that is assumed to be virtually present for executing the IL code
Locals	number and information about the local variables in the method; here it is variable srt is of short type (i.e. System.Int16) and it also has another temporary variable used for return value (the local variables are indexed from zero).
IL_xxxx	refers to the label names for the IL instructions

Pop Quiz

What is the **.maxstack** size value for the expression **“(1 - (2 / 3)) + ((4 % 5) * 6)”?**

1 2 3 / - 4 5 % 6 * +



Answer:
.maxstack value is 3

Pop Quiz

Guess what the instruction **ldc.i4.m1** stand for?

Answer

ldc.i4.m1 stands for load constant of int32 type with value **-1** on to the execution stack

Pop Quiz

```
// method line 2
.method private static hidebysig
           default void Main ()  cil managed
{
        // Method begins at RVA 0x2058
  .entrypoint
  // Code size 21 (0x15)
  .maxstack 3
  .locals init (
    int32  V_0)
  IL_0000:  ldc.i4.0
  IL_0001:  stloc.0
  IL_0002:  ldloc.0
  IL_0003:  dup
  IL_0004:  ldc.i4.1
  IL_0005:  add
  IL_0006:  stloc.0
  IL_0007:  call void class [mscorlib]System.Console::WriteLine(int32)
  IL_000c:  ldloc.0
  IL_000d:  ldc.i4.s 0x0a
  IL_000f:  blt IL_0002

  IL_0014:  ret
} // end of method Hello::Main
```

Decompile this assembly code

Answer

```
// method line 2
.method private static hidebysig
           default void Main ()  cil managed
{
   // Method begins at RVA 0x2058
  .entrypoint
  // Code size 21 (0x15)
  .maxstack 3
  .locals init (
    int32  V_0)
  IL_0000:  ldc.i4.0
  IL_0001:  stloc.0
  IL_0002:  ldloc.0
  IL_0003:  dup
  IL_0004:  ldc.i4.1
  IL_0005:  add
  IL_0006:  stloc.0
  IL_0007:  call void class [mscorlib]System.Console::WriteLine(int32)
  IL_000c:  ldloc.0
  IL_000d:  ldc.i4.s 0x0a
  IL_000f:  blt IL_0002

  IL_0014:  ret
} // end of method Hello::Main
```

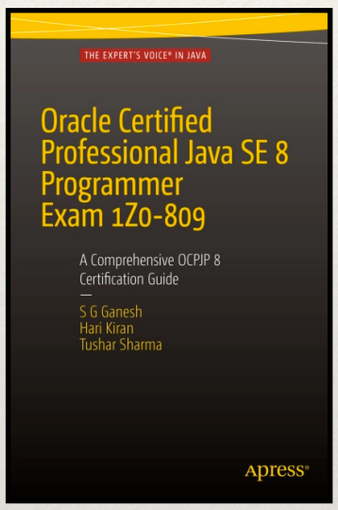
```
static void Main()
{
    int i = 0;
    do {
        Console.WriteLine(i++);
    }
    while(i < 10);
}
```

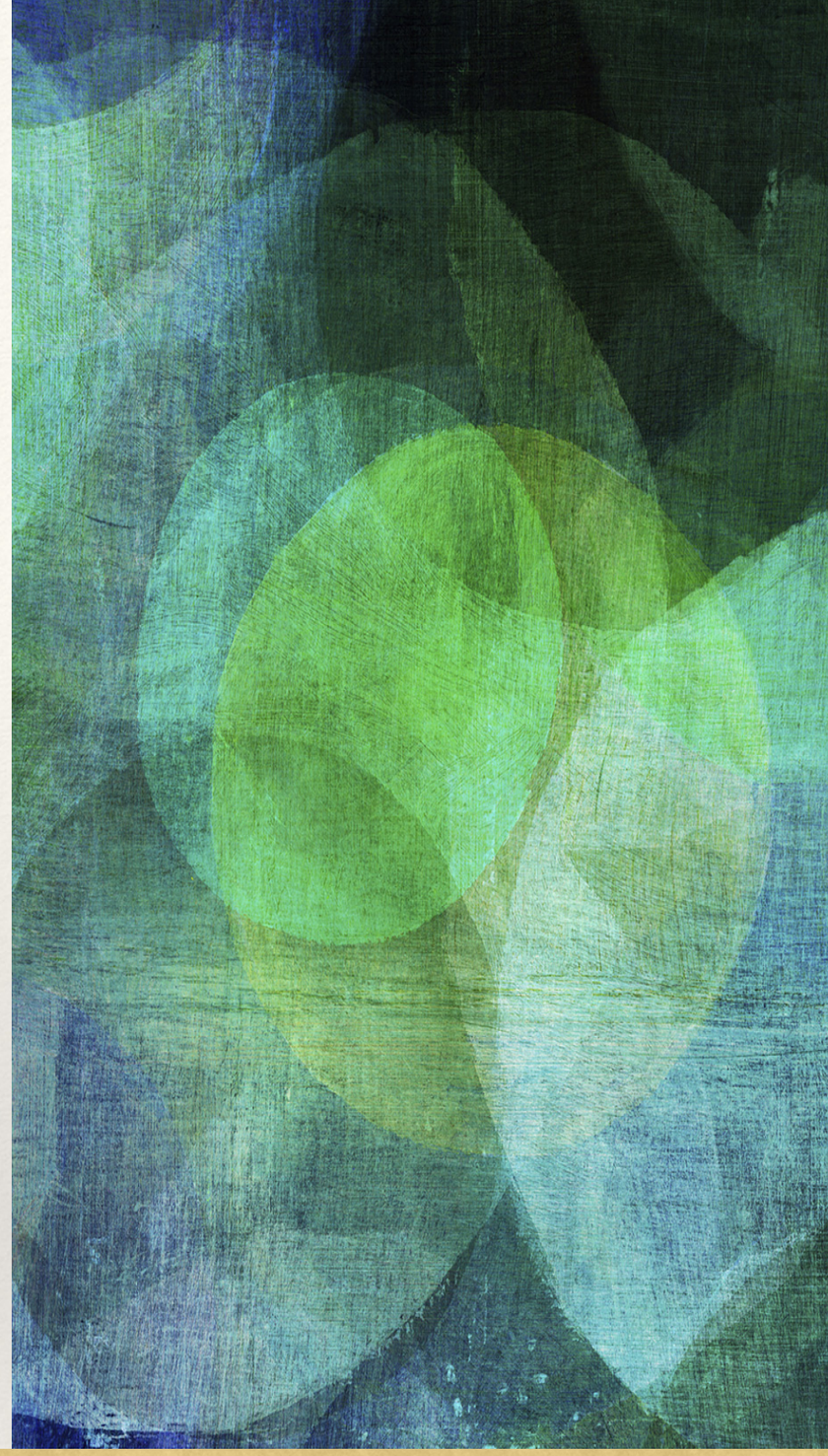
TO READ

- "Inside .NET" article - http://www.slideshare.net/sgganesh/insidenet
- "An overview of MSIL" - http://www.slideshare.net/sgganesh/overview-of-msil
- "Intermediate languages" - http://www.slideshare.net/sgganesh/intermediate-languages
- "Common Language Specification (CLS)" - https://msdn.microsoft.com/library/12a7a7h3(v=vs.100).aspx
- "Common Intermediate Language" - https://en.wikipedia.org/wiki/Common_Intermediate_Language
- "List of CIL instructions" - https://en.wikipedia.org/wiki/List_of_CIL_instructions
- "ECMA C# and Common Language Infrastructure Standards" - https://www.visualstudio.com/en-us/mt639507

IMAGE CREDITS

- https://pixabay.com/static/uploads/photo/2015/12/28/15/58/ferrari-1111582_960_720.jpg
- http://i.dailymail.co.uk/i/pix/2014/08/29/article-0-0296355F000004B0-113_634x421.jpg
- http://blogs.shell.com/climatechange/wp-content/uploads/2015/01/Check-under-the-hood.jpg
- https://diaryofabusymumdotcom.files.wordpress.com/2015/01/1369952540_be029c8337.jpg
- http://trentarthur.ca/wp-content/uploads/2013/05/gatsby.jpg
- http://cdn.playbuzz.com/cdn/84b94651-08da-4191-9b45-069535cf523f/9c35f887-a6fc-4c8d-861a-f323078709e8.jpg
- http://pad2.whstatic.com/images/thumb/5/54/Draw-a-Simple-Tree-Step-2.jpg/aid594851-728px-Draw-a-Simple-Tree-Step-2.jpg
- http://www.seabreeze.com.au/Img/Photos/Windsurfing/5350271.jpg
- https://d.gr-assets.com/hostedimages/1380222758ra/461081.gif
- http://cdn.shopify.com/s/files/1/0021/6982/products/GW-7693274_large.jpg?v=1283553128
- http://www.fisher-price.com/en_IN/Images/RMA_RWD_rock_a_stack_tcm222-163387.jpg
- http://www.njfamily.com/NJ-Family/January-2011/Learn-How-to-Spot-a-Learning-Disability/Boy-learning-disability.jpg
- https://teens.drugabuse.gov/sites/default/files/styles/medium/public/NIDA-News-What-was-down-the-hole-Alice.jpg?itok=DH19L7F2
- http://archivedemo.cnx.org/resources/4df9b85136bb00ee04456b031aa0c344e54f282e/CNX_Psych_08_04_Knuckles.jpg
- http://archivedemo.cnx.org/resources/4df9b85136bb00ee04456b031aa0c344e54f282e/CNX_Psych_08_04_Knuckles.jpg
- http://www.urbanspaces.co.uk/image/error-message-error-us.jpg
- http://conservationmagazine.org/wordpress/wp-content/uploads/2013/05/dig-deeper.jpg
- http://4.bp.blogspot.com/-BAZm9rddEhQ/TWy441M-p1I/AAAAAAAAAQg/_SKF8PMkVHA/s1600/mr%2Bfixit.tif%2B%2528Converted%2529--6.jpg





ganesh@codeops.tech

www.codeops.tech

+91 98801 64463

@GSamarthyam

slideshare.net/sgganesh

bit.ly/sgganesh

CodeOps