

# Many High-Level Languages

Choosing a programming language...

- ▶ Basic
- ▶ Fortran 95
- ▶ Ada
- ▶ Python
- ▶ Haskell
- ▶ C++
- ▶ Java

We choose Java because it has fewer ways “to shoot yourself in the foot.”

# Terminology

- ▶ compiling (originally): linking subroutines
- ▶ translating: converting from a high-level language to a low-level
- ▶ compiling: translating to native (of some real machine) code
- ▶ batch: compile once, run many times
- ▶ interpreting: running the program under the control of a software program
- ▶ interactive: read-eval-print loop. Evaluate means interpret in this case; but now it could be extended to compile and run.

# Implementation

A language may be translated or implemented by zero, one, or many different systems.

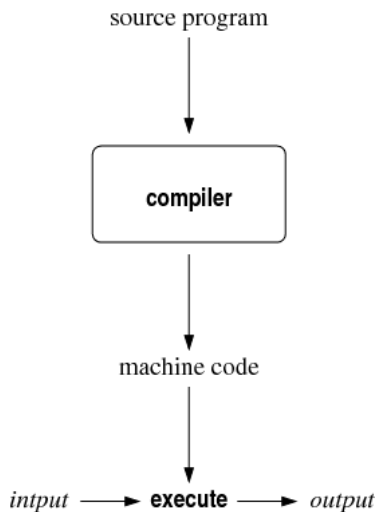
Unqualified statements such as

1. “Language *X* is compiled.”
2. “Language *X* is slow.”

are nonsense, because the speed of execution and the type of translation depend on the implementation. Of course, the language may influence or seek to influence the implementation. A language may be closely associated with a particular implementation. But a language itself is not an implementation.

The world of language implementations is often quite complex. For example, there are the GNU `gfortran` and `g77` compilers, not to mention many commercial compilers for Fortran. Also, there are the SUN JDK tools for Java, the IBM Jikes compiler for Java, and the GNU `gjc` compiler.

# Traditional Compilation



How do you solve a hard problem?

## How do you solve a hard problem?

One important approach is to break it into well-defined subproblems.  
(A compiler is just a big program.)

# Compilation Steps

When examined in more detail, compilation takes several steps.

1. preprocessing, macro processing
2. translation (compiling)
3. assembling mnemonics
4. linking other code and preparing for execution

Macros (an extremely dangerous facility) are common in C and C++. Java does preprocessing to translate character sets.

# Language Systems

Language translation and execution systems are big and complex these days because computers can execute larger and larger programs faster and faster. The programmer or program user rarely sees the individual steps.

IDEs, interactive language systems, JIT compilers, incremental compilers, and dynamic linking all conspire to hide and blur the important individual steps. (But make programming development faster and easier).

# Interpreting

An *interpreter* is a program that takes another program as input and executes it, possibly line-by-line, possibly without translating it to an intermediate form. Sometimes the translation is to an intermediate form which may be executed by a *virtual* or *abstract machine*.

Examples of abstract machines include: Forth virtual machine, p-code machine (Pascal), Python virtual machine, SECD machine (lambda calculus), Smalltalk virtual machine, Warren Abstract machine (Prolog).

As hardware gets faster, the advantage of portability overtakes the disadvantage of slow emulation, and multi-language virtual machines are becoming more important: the Microsoft .Net platform (C#, Managed C++, Python) and the Java virtual machine (Java, Jython, Ada). Since these abstract machines execute complex source languages the machines must also provide the runtime support these languages expect.

## Interpreting (continued)

Since an abstract machine may be abstract by virtue of having abstract instructions *or* by having abstract capabilities, the term abstract/virtual machine may be ambiguous and lead to confusion. Abstract instructions are likely to be slower than real instructions because of the extra software overhead of interpretation. Abstract capabilities are likely to be faster than programmer-supplied code because of the skill of the implementors and the use of the underlying machine.

The key aspect of an interpreter is emulation. The key aspect of a runtime system is support of functionality.

# Runtime system

Modern, high-level languages require that a program have additional support during execution. This is sometimes called the runtime system. The runtime system contains lots of code that is not written by the programmer, but was written by others and used when a program in the language is run.

The runtime system may provide support for mathematical operations (e.g., exponentiation), floating-point arithmetic, complex numbers, high-level input and output functions, concurrency, memory management (e.g., garbage collection), etc. Modern languages tend to have larger and larger support systems.

The work of the runtime system may require assistance of the translation system, for example, to insert reference counting code, debugging code, etc. The runtime system must be available to every program in the language so it can run correctly, but none of the functionality might actually be used.

# Runtime system

The distinction between the runtime system and the standard libraries is not always clear. Take these two statements in Java:

```
System.out.printf ("%d %s", 4, this);  
new Thread ().start ();
```

Both statements appear to be just simple calls to library routines, but ultimately considerable code gets executed which the programmer did not, could not, or would not write (in Java).

Back to translation ...

# Important Unix Tools



- ▶ gcc
- ▶ gas
- ▶ gdb
- ▶ make
- ▶ objdump
- ▶ uname
- ▶ od

## Compilation — gcc

```
#include <stdio.h>
```

```
int
```

```
main () {
```

```
    fputs ("Hello world!\n", stdout);
```

```
    return 0;
```

```
}
```

# Compilation — gcc

```
%gcc -o hello -v hello.c
Reading specs from /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.6/2.95.3/specs
gcc version 2.95.3 20010315 (release)
  /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.6/2.95.3/cpp0
    -lang-c -v -D__GNUC__=2 -D__GNUC_MINOR__=95 -Dsparc -Dsun -Dunix -D__svr4__ -D__SVR4 -D__sparc__ -D__sun__
    -Asystem(unix) -Asystem(svr4) -D__GCC_NEW_VARARGS__ -Acpu(sparc) -Amachine(sparc) hello.c /var/tmp/cc5V4Wyl
GNU CPP version 2.95.3 20010315 (release) (sparc)
#include "... " search starts here:
#include <...> search starts here:
  /software/solaris/gnu/include
  /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.6/2.95.3/../../../../sparc-sun-solaris2.6/include
  /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.6/2.95.3/include
  /usr/include
End of search list.
The following default directories have been omitted from the search path:
  /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.6/2.95.3/../../../../include/g++-3
End of omitted list.
  /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.6/2.95.3/ccl
    /var/tmp/cc5V4Wyl.i -quiet -dumpbase hello.c -version -o /var/tmp/cc47fQVU.s
GNU C version 2.95.3 20010315 (release) (sparc-sun-solaris2.6) compiled by GNU C version 3.0.3.
  /software/solaris/gnu/bin/as -V -Qy -s -o /var/tmp/ccNHrBWS.o /var/tmp/cc47fQVU.s
GNU assembler version 2.11.2 (sparc-sun-solaris2.6) using BFD version 2.11.2
  /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.6/2.95.3/collect2
    -v -Y P,/usr/ccs/lib:/usr/lib -Qy -o hello /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.6/2.95.3/cr
GNU ld version 2.11.2 (with BFD 2.11.2)
Supported emulations:
  elf32_sparc
```

```
cs> gcc -S hello.c -o hello.s
```

```
cs> gcc -S hello.c -o hello.s
```

```
        .file      "hello.c"
gcc2_compiled.:
.section .rodata
        .align 8
.LLC0:
        .asciz    "Hello world!\n"
.section  ".text"
        .align 4
        .global  main
        .type    main,#function
        .proc    04
main:
        !#PROLOGUE# 0
        save     %sp, -112, %sp
        !#PROLOGUE# 1
        sethi   %hi(.LLC0), %o1
        or      %o1, %lo(.LLC0), %o0
        sethi   %hi(__iob+16), %o2
        or      %o2, %lo(__iob+16), %o1
        call    fputs, 0
        nop
        mov     0, %i0
        b       .LL2
        nop
```

**gcc compiles C to native code**

# Compilation

0000 7f45 4c46 0102 0100 0000 0000 0000 0000	del E L F soh stx soh nul nul nul nul
0020 0001 0002 0000 0001 0000 0000 0000 0000	nul soh nul stx nul nul nul soh nul nul
0040 0000 00e8 0000 0000 0000 0034 0000 0000 0028	nul nul nul h nul nul nul nul nul
4 nul nul nul nul nul (	
0060 000a 0007 9de3 bf90 1300 0000 9012 6000	nul nl nul bel gs c ? dle dc3 nul nul
` nul	
0100 1500 0000 9212 a000 4000 0000 0100 0000	nak nul nul nul dc2 dc2 sp nul @ nul nul
0120 b010 2000 1080 0002 0100 0000 81c7 e008	0 dle sp nul dle nul nul stx soh nul nul
G ` bs	
0140 81e8 0000 0000 0000 4865 6c6c 6f20 776f	soh h nul nul nul nul nul nul H
e l l o sp w o	
0160 726c 6421 0a00 0000 0047 4343 3a20 2847	r l d ! nl nul nul nul nul
G C C : sp ( G	
0200 4e55 2920 322e 3935 2e33 2032 3030 3130	N U ) sp 2 . 9 5 .
3 sp 2 0 0 l 0	
0220 3331 3520 2872 656c 6561 7365 2900 002e	3 l 5 sp ( r e l e
a s e ) nul nul .	
0240 7379 6d74 6162 002e 7374 7274 6162 002e	s y m t a b nul . s
t r t a b nul .	
0260 7368 7374 7274 6162 002e 7465 7874 002e	s h s t r t a b nul
. t e x t nul .	
0300 7265 6c61 2e74 6578 7400 2e64 6174 6100	r e l a . t e x t nul
. d a t a nul	
0320 2e62 7373 002e 726f 6461 7461 002e 636f	. b s s nul . r o d
a t a nul c o	
0340 6d6d 656e 7400 0000 0000 0000 0000 0000	m m e n t nul nul nul nul nul nul
0360 0000 0000 0000 0000 0000 0000 0000 0000	nul nul nul nul nul nul nul nul nul nul
*	
0420 0000 001b 0000 0001 0000 0006 0000 0000	nul nul nul esc nul nul nul soh nul nul nul
0440 0000 0034 0000 0030 0000 0000 0000 0000	nul nul nul 4 nul nul nul 0 nul nul nul
0460 0000 0004 0000 0000 0000 0021 0000 0004	nul nul nul eot nul nul nul nul nul nul
! nul nul nul eot	
0500 0000 0000 0000 0000 0000 0368 0000 003c	nul nul nul nul nul nul nul nul nul nul etx
h nul nul nul <	
0520 0000 0008 0000 0001 0000 0004 0000 000c	nul nul nul bs nul nul nul soh nul nul nul
ff	
0540 0000 002c 0000 0001 0000 0003 0000 0000	nul nul nul , nul nul nul soh nul nul nul
0560 0000 0064 0000 0000 0000 0000 0000 0000	nul nul nul d nul nul nul nul nul nul nul
0600 0000 0001 0000 0000 0000 0032 0000 0008	nul nul nul soh nul nul nul nul nul nul nul
2 nul nul nul bs	

# Compilation

```
hello.o:      file format elf32-sparc
```

```
Contents of section .text:
```

```
0000 9de3bf90 13000000 90126000 15000000 .....`.....
0010 9212a000 40000000 01000000 b0102000 ....@..... .
0020 10800002 01000000 81c7e008 81e80000 .....
```

```
Contents of section .data:
```

```
Contents of section .rodata:
```

```
0000 48656c6c 6f20776f 726c6421 0a000000 Hello world!....
```

```
Contents of section .comment:
```

```
0000 00474343 3a202847 4e552920 322e3935 .GCC: (GNU) 2.95
0010 2e332032 30303130 33313520 2872656c .3 20010315 (rel
0020 65617365 2900                                     ease).
```

```
Disassembly of section .text:
```

```
00000000 <main>:
```

```
0:  9d e3 bf 90      save %sp, -112, %sp
4:  13 00 00 00      sethi %hi(0), %o1
8:  90 12 60 00      mov %o1, %o0 ! 0 <main>
c:  15 00 00 00      sethi %hi(0), %o2
10: 92 12 a0 00      mov %o2, %o1 ! 0 <main>
14: 40 00 00 00      call 14 <main+0x14>
18: 01 00 00 00      nop
1c: b0 10 20 00      clr %i0 ! 0 <main>
20: 10 80 00 02      b 28 <main+0x28>
24: 01 00 00 00      nop
28: 81 c7 e0 08      ret
2c: 81 e8 00 00      restore
```

# ELF – Executable and Linkable Format

Linking View

ELF header
Program header table (optional)
section 1
...
section n
...
...
Section header table

Execution View

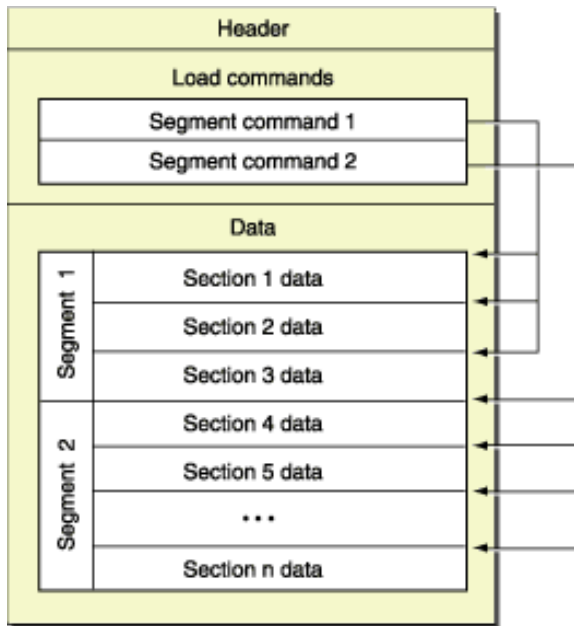
ELF header
Program header table
Segment 1
Segment 2
...
...
Section header table (optional)

# ELF – Executable and Linkable Format

```
typedef struct {
    unsigned char e_ident[16]; /* version and other info */
    uint16_t      e_type;      /* none, relocatable, executable, shared, core */
    uint16_t      e_machine;   /* none, SPARC, Intel, Motorola, MIPS, ... */
    uint32_t      e_version;
    uintN_t       e_entry;     /* entry point */
    ...
} ElfN_Ehdr;
```

**Note** `readelf` (Unix) and `elfdump` (Solaris) view elf files. Note `otool` (Darwin) to view Mach-o files.

# Mach-O



# Mach-O

## (Pronounced “macho.”)

```
/* From #include <mach-o/loader.h> */
/* Mach header of the object file for 32-bit architectures. */
struct mach_header {
    uint32_t    magic;           /* mach magic number identifier */
    cpu_type_t  cputype;        /* PowerPC, I386 */
    cpu_subtype_t cpusubtype;   /* machine specifier */
    uint32_t    filetype;       /* object, executable, shared, core, ... */
    uint32_t    ncmds;          /* number of load commands */
    uint32_t    sizeofcmds;     /* the size of all the load commands */
    uint32_t    flags;          /* flags */
};

/* Constant for the magic field of the mach_header (32-bit architectures) */
#define MH_MAGIC    0xfeedface /* the mach magic number */
#define MH_CIGAM   0xcefaedfe /* NXSwapInt(MH_MAGIC) */
```

# Translating Java

A wide range of techniques are used in translating Java into executable form. Several translators exist for the language.

1. IBM jikes
2. GNU gcj
3. Sun Java 2 SDK

# Translating Java

A wide range of techniques are used in translating Java into executable form. Several translators exist for the language.

1. IBM jikes
2. GNU gcj
3. Sun Java 2 SDK

We begin by looking at GNU gcj to see a traditional translator in action. Then we move to the SUN Java 2 SDK and see the important role of byte code.

## Compilation — gcj

```
public class Hello {  
  
    public static void main (String[] args) {  
        System.out.println ("Hello world!");  
    }  
  
}
```

# Compilation — gcj

```
Reading specs from /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.9/3.3.2/specs
Reading specs from /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.9/3.3.2/../../../../libgcj.spec
rename spec lib to liborig
Configured with: ./configure --prefix=/software/solaris/gnu --with-ld=/software/solaris/gnu/bin/ld --with-as=
Thread model: posix
gcc version 3.3.2
  /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.9/3.3.2/jc1 Hello.java -fuse-divide-subroutine -fcheck-r
GNU Java version 3.3.2 (sparc-sun-solaris2.9)
  compiled by GNU C version 2.95.3 20010315 (release).
GGC heuristics: --param ggc-min-expand=47 --param ggc-min-heapsize=32768
Class path starts here:
  ./
  /software/solaris/gnu/share/java/libgcj-3.3.2.jar/ (system) (zip)
  /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.9/3.3.2/../../../../sparc-sun-solaris2.9/bin/as -V -Qy -
GNU assembler version 2.14 (sparc-sun-solaris2.9) using BFD version 2.14 20030612
  /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.9/3.3.2/jvgenmain Hellomain /var/tmp//ccWJ2hCQ.i
  /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.9/3.3.2/ccl /var/tmp//ccWJ2hCQ.i -quiet -dumpbase Hellom
GNU C version 3.3.2 (sparc-sun-solaris2.9)
  compiled by GNU C version 2.95.3 20010315 (release).
GGC heuristics: --param ggc-min-expand=47 --param ggc-min-heapsize=32768
  /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.9/3.3.2/../../../../sparc-sun-solaris2.9/bin/as -V -Qy -
GNU assembler version 2.14 (sparc-sun-solaris2.9) using BFD version 2.14 20030612
  /software/solaris/gnu/lib/gcc-lib/sparc-sun-solaris2.9/3.3.2/collect2 -V -Y P,/usr/ccs/lib:/usr/lib -Qy -o he
GNU ld version 2.14 20030612
Supported emulations:
  elf32_sparc
  elf64_sparc
```

```
cs> gcj -S Hello.java -o hello.s
```

```
cs> gcj -S Hello.java -o hello.s
```

*gcj compiles Java  
to native code*

```
_ZN5Hello4mainEP6JArrayIPN4java4lang6StringE)
    !#PROLOGUE# 0
    save    %sp, -128, %sp
.LLCFI0:
    !#PROLOGUE# 1
    st      %i0, [%fp+68]
.LLBB2:
    sethi   %hi(_ZN4java4lang6System6class$E), %g1
    or      %g1, %lo(_ZN4java4lang6System6class$E),
    mov     1, %o4
    stb     %o4, [%fp-18]
    ldub    [%g1+90], %g1
    sll     %g1, 24, %g1
    sra     %g1, 24, %g1
    cmp     %g1, 14
    bge     .LL2
    nop
    ...
```

Same kind of assembler output, ELF file, etc, etc.

## Translation — Sun JDK

There are two translation tools in the Sun JDK.

javac java

compiler? JVM

## Translation — Sun JDK

Same program again.

```
public class HelloWorld {  
  
    public static void main (String args[]) {  
        System.out.println ("Hello World!");  
    }  
  
}
```

# Translation — Sun JDK

The output of the `javac` is a binary file known as a class file. This file contains the programming instructions in what is known as byte code.

```
000 ca fe ba be 00 00 00 31 00 1a 0a 00 06 00 0c 09 00 0d
018 00 0e 08 00 0f 0a 00 10 00 11 07 00 12 07 00 13 01 00
036 06 3c 69 6e 69 74 3e 01 00 03 28 29 56 01 00 04 43 6f
054 64 65 01 00 04 6d 61 69 6e 01 00 16 28 5b 4c 6a 61 76
072 61 2f 6c 61 6e 67 2f 53 74 72 69 6e 67 3b 29 56 0c 00
090 07 00 08 07 00 14 0c 00 15 00 16 01 00 0c 48 65 6c 6c
108 6f 20 57 6f 72 6c 64 21 07 00 17 0c 00 18 00 19 01 00
126 0a 48 65 6c 6c 6f 57 6f 72 6c 64 01 00 10 6a 61 76 61
144 2f 6c 61 6e 67 2f 4f 62 6a 65 63 74 01 00 10 6a 61 76
162 61 2f 6c 61 6e 67 2f 53 79 73 74 65 6d 01 00 03 6f 75
180 74 01 00 15 4c 6a 61 76 61 2f 69 6f 2f 50 72 69 6e 74
198 53 74 72 65 61 6d 3b 01 00 13 6a 61 76 61 2f 69 6f 2f
216 50 72 69 6e 74 53 74 72 65 61 6d 01 00 07 70 72 69 6e
234 74 6c 6e 01 00 15 28 4c 6a 61 76 61 2f 6c 61 6e 67 2f
252 53 74 72 69 6e 67 3b 29 56 00 20 00 05 00 06 00 00 00
270 00 00 02 00 00 00 07 00 08 00 01 00 09 00 00 00 11 00
288 01 00 01 00 00 00 05 2a b7 00 01 b1 00 00 00 00 09
306 00 0a 00 0b 00 01 00 09 00 00 00 15 00 02 00 01 00 00
324 00 09 b2 00 02 12 03 b6 00 04 b1 00 00 00 00 00 00
```

# Translation — Sun JDK

You can convert a class back to mnemonics to get an idea of what information is in the class file.

```
> javap -c HelloWorld
```

```
class HelloWorld extends java.lang.Object {
HelloWorld();
  0:  aload_0
  1:  invokespecial  #1; //Method java/lang/Object."<init>":()V
  4:  return

public static void main(java.lang.String[]);
  0:  getstatic     #2; //Field java/lang/System.out:Ljava/io/PrintStream;
  3:  ldc          #3; //String Hello World!
  5:  invokevirtual #4; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
  8:  return
}
```

## Java – Just-In-Time

Although Java is can be interpreted by a Java virtual machine (JVM), the byte-code could be compiled to native code. An independent, executable file may or may not be made. It is even possible to compile only some of the byte-code—the parts that are executed a lot—and not other parts.

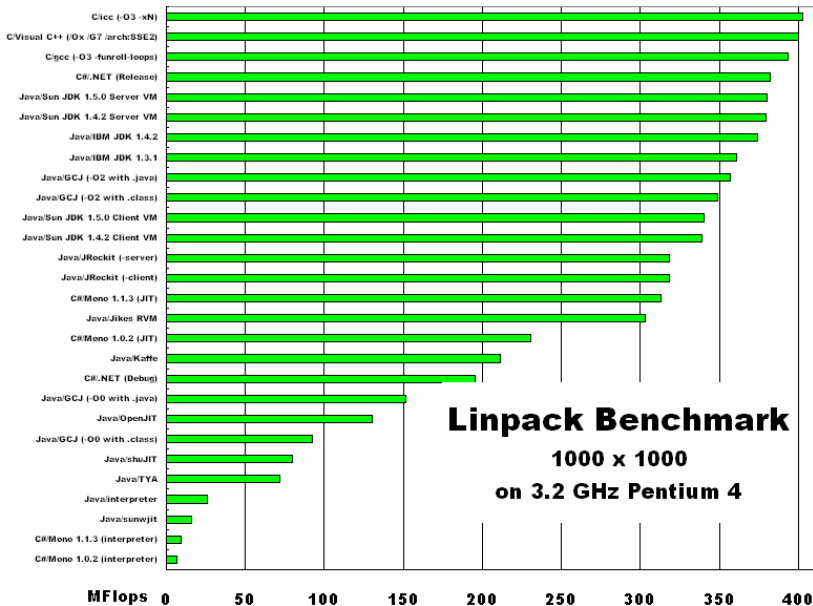
Sun Microsystems calls the program `java` a “launcher” as details of the actions differ from typical compilers or interpreters. Such a translation/execution system is called a just-in-time (JIT) compiler, and may only compile parts of the byte-code when (and if) they are reached or executed often.

If you want Java to interpret the byte-code, you must ask for it:

```
cs> java -Xint Main
```

Do not confuse a language with its implementation.

Benchmarks mean very little.



1.0	C++ GNU g++	1.35
1.7	Java 6 -server	2.29
1.7	C GNU gcc	2.31
2.3	Haskell GHC	3.14
2.7	Intel Fortran	3.71
2.8	Pascal Free Pascal	3.74
3.3	C# Mono	4.44
3.8	Ada 2005 GNAT	5.09
12	Java 6 -Xint	16.03
17	Smalltalk VisualWorks	23.12
26	Python	35.43
33	Mozart/Oz	44.62
44	Perl	59.81
51	PHP	68.79
77	Ruby	104.01

Computer Language Benchmarks Game. January 2009. Platform: Ubuntu, 2.4Ghz Intel Q6600 quad-core. First number is ratio to GNU C++ of the third column: geometric mean of the measure for the language to the best measurement for any language over all 11 benchmarks.