# Language Processors

Stephen A. Edwards

Columbia University

Fall 2013

Interpreter

```
                    ┌─────────────────┐
                    │ Source Program  │
                    └─────────────────┘
                             │
                             ▼
┌─────────┐          ┌─────────────────┐          ┌─────────┐
│  Input  │─────────▶│   Interpreter   │─────────▶│ Output  │
└─────────┘          └─────────────────┘          └─────────┘
```
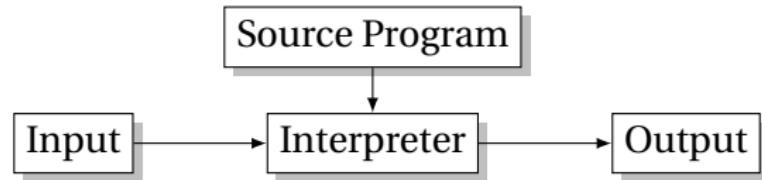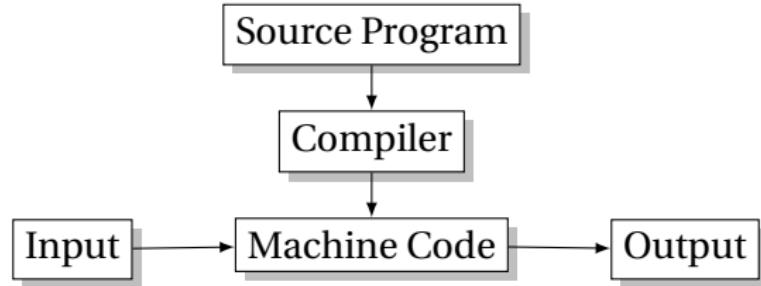
Compiler

Bytecode Interpreter
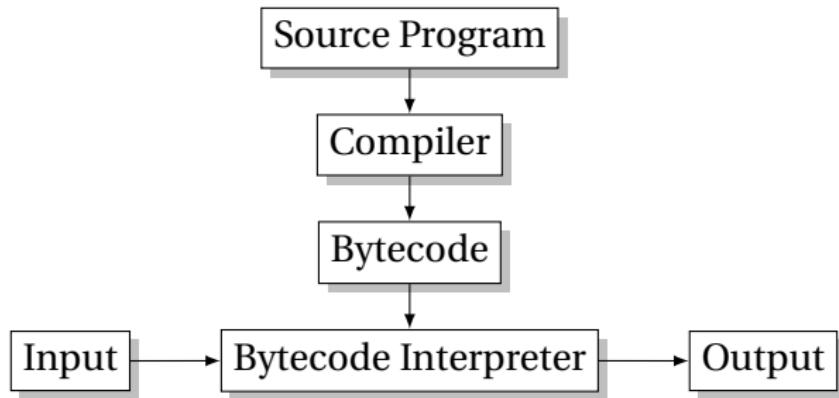
Just-In-Time Compiler
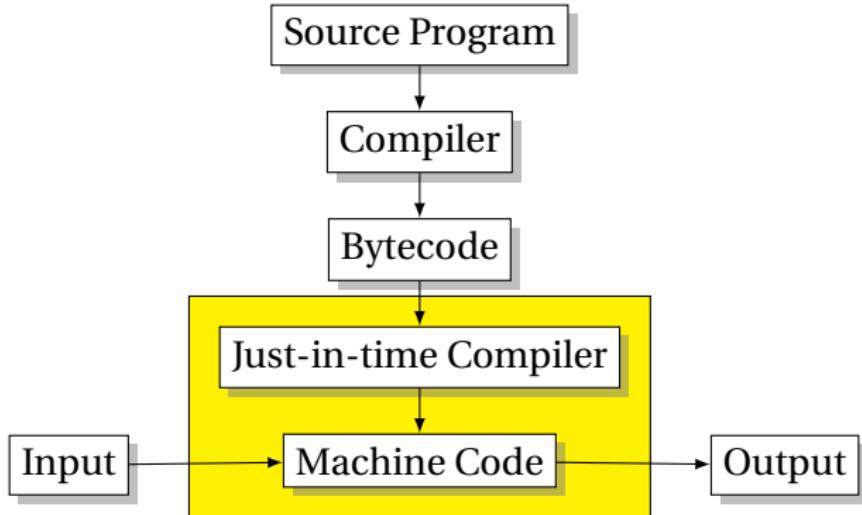
Language Speeds Compared

Native code compilers

Just-in-time compilers

Bytecode interpreters

ATS
C++ GNU g++
C GNU gcc
Java 6 steady state
Ada 2005 GNAT
Haskell GHC
Scala
Java 6 -server
Lua LuaJIT
Fortran Intel
Clean
OCaml
F# Mono
C# Mono
Pascal Free Pascal
Go 6g 8g
Racket
Lisp SBCL
JavaScript V8
Erlang HiPE
Lua
Smalltalk VisualWorks
Java 6 -Xint
Python CPython
Python 3
Ruby 1.9
Mozart/Oz
Ruby JRuby
PHP
Perl

Source: http://shootout.alioth.debian.org/

Separate Compilation

Preprocessor

"Massages" the input before the compiler sees it.

- Macro expansion
- File inclusion
- Conditional compilation

The C Preprocessor

```
#include <stdio.h>
#define min(x, y) \
  ((x)<(y))?(x):(y)
#ifdef DEFINE_BAZ
int baz();
#endif
void foo()
{
  int a = 1;
  int b = 2;
  int c;
  c = min(a,b);
}
```

```
extern int printf(char*,...);
/* ... many more declarations
from stdio.h */


void foo()
{
  int a = 1;
  int b = 2;
  int c;
  c = ((a)<(b))?(a):(b);
}
```

Compiling a Simple Program

```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

What the Compiler Sees

```c
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

```
i  n  t  sp  g  c  d  (  i  n  t  sp  a  ,  sp  i
n  t  sp  b  )  nl  {  nl  sp  sp  w  h  i  l  e  sp
(  a  sp  !  =  sp  b  )  sp  {  nl  sp  sp  sp  sp  i
f  sp  (  a  sp  >  sp  b  )  sp  a  sp  -  =  sp  b
;  nl  sp  sp  sp  sp  e  l  s  e  sp  b  sp  -  =  sp
a  ;  nl  sp  sp  }  nl  sp  sp  r  e  t  u  r  n  sp
a  ;  nl  }  nl
```

Text file is a sequence of characters

# Lexical Analysis Gives Tokens

```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```


int	gcd	(	int	a	,	int	b	)	{	while	(	a
!=	b	)	{	if	(	a	>	b	)	a	-=	b	;	else
b	-=	a	;	}	return	a	;	}
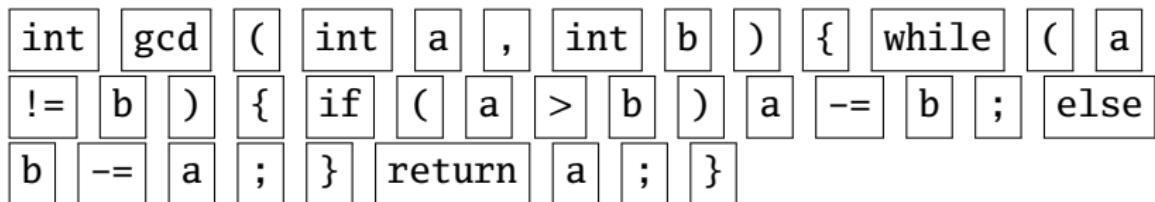
A stream of tokens. Whitespace, comments removed.

Parsing Gives an Abstract Syntax Tree
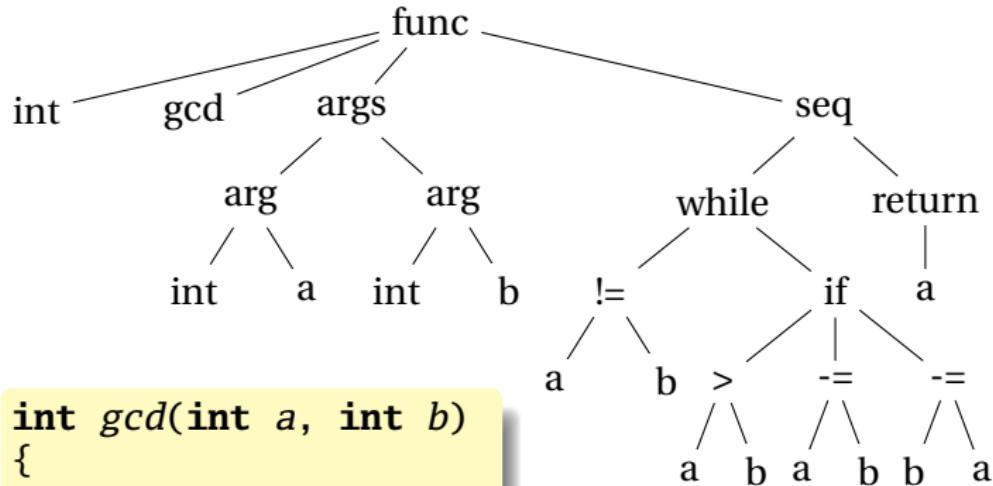

```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

Semantic Analysis Resolves Symbols and Checks Types
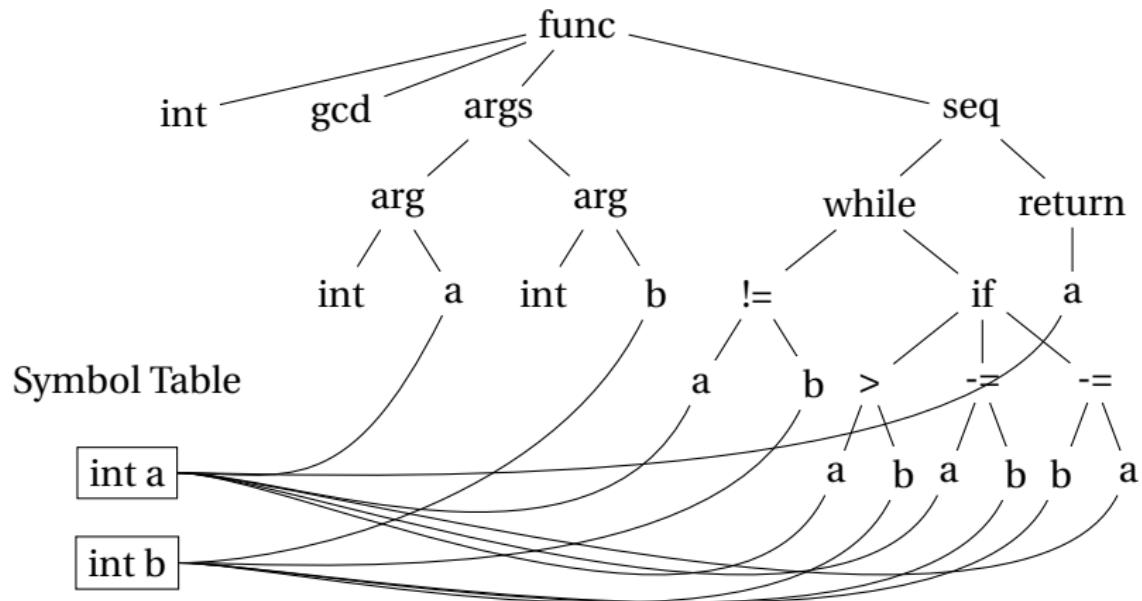
Translation into 3-Address Code

```
L0: sne   $1,  a, b
    seq   $0, $1, 0
    btrue $0, L1    # while (a != b)
    sl    $3,  b, a
    seq   $2, $3, 0
    btrue $2, L4    # if (a < b)
    sub   a,   a, b # a -= b
    jmp   L5
L4: sub   b,   b, a # b -= a
L5: jmp   L0
L1: ret     a
```

```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

Idealized assembly language w/ infinite registers

Generation of 80386 Assembly
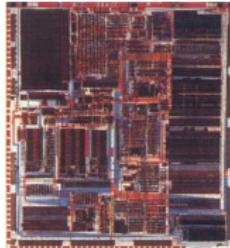

```
gcd:    pushl %ebp              # Save BP
        movl  %esp,%ebp
        movl  8(%ebp),%eax      # Load a from stack
        movl  12(%ebp),%edx     # Load b from stack
.L8:    cmpl  %edx,%eax
        je    .L3               # while (a != b)
        jle   .L5               # if (a < b)
        subl  %edx,%eax         # a -= b
        jmp   .L8
.L5:    subl  %eax,%edx         # b -= a
        jmp   .L8
.L3:    leave                   # Restore SP, BP
        ret
```