**What are Compilers?**

- Translators from one representation of program code to another
- Typically: high-level source code to machine language (object code)
- Not always:
  - Java compiler: Java to interpretable bytecodes
  - Java JIT: bytecode to executable image
Source Code

- Source code: optimized for human readability
  - expressive: matches human notions of grammar
  - redundant to help avoid programming errors
  - computation possibly not fully determined by code

```c
int expr(int n)
{
    int d;
    d = 4 * n * n * (n + 1) * (n + 1);
    return d;
}
```

Machine Code

- Optimized for hardware
  - Redundancy, ambiguity reduced
  - Information about intent and reasoning lost
  - Assembly code ≈ machine code

```
expr:
pushl %ebp  55
movl %esp, %ebp  89 e5
subl $4, %esp  83 ec 04
movl 8(%ebp), %eax  8b 45 08
movl %eax, %edx  89 c2
imull 8(%ebp), %edx  0f af 55 08
movl 8(%ebp), %eax  8b 45 08
incl %eax  40
imull %eax, %edx  0f af d0
movl 8(%ebp), %eax  8b 45 08
incl %eax  40
imull %edx, %eax  0f af c2
sall $2, %eax  c1 e6 02
movl %eax, -4(%ebp)  89 45 fc
movl -4(%ebp), %eax  c9 45 fc
leave
ret
```

Example (Output assembly code)

Unoptimized Code

```
expr:
pushl %ebp
movl %esp, %ebp
subl $4, %esp
movl 8(%ebp), %eax
movl %eax, %edx
imull 8(%ebp), %edx
movl 8(%ebp), %eax
incl %eax
imull %eax, %edx
movl 8(%ebp), %eax
movl -4(%ebp), %eax
leave
ret
```

Optimized Code

```
expr:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %edx
movl %edx, %eax
imull %edx, %eax
incl %edx
imull %edx, %eax
imull %edx, %eax
sall $2, %eax
leave
ret
```

How to translate?

- Source code and machine code mismatch
- Goals:
  - source-level expressiveness for task
  - best performance for concrete computation
  - reasonable translation efficiency (< O(n^3))
  - maintainable compiler code
**How to translate correctly?**

- Programming languages describe computation precisely
- Therefore: translation can be precisely described (a compiler can be correct)
- Correctness is very important!
  - hard to debug programs with broken compiler...
  - non-trivial: programming languages are expressive
  - implications for development cost, security
  - some compilers have been **proven** correct!

[X. Leroy, *Formal Verification of a Realistic Compiler*, CACM 52(7), 2009]

---

**How to translate effectively?**

![Diagram](image-url)

- High-level source code
- ?
- Low-level machine code
Idea: translate in steps

- Compiler uses a series of different intermediate representations (IRs) of programs.
- Different IRs are good for different phases of compilation.

Compilation in a Nutshell 1

Source code (character stream)

if (b == 0) a = b;

Lexical analysis

Token stream

if ( b == 0 ) a = b ;

Parsing

Abstract syntax tree (AST)

int b
==
0
a
b

Semantic Analysis

Decorated AST

boolean
==
int
int b
int 0
int a int
lvalue
Compilation in a Nutshell 2

if b == 0 goto L1 else L2
L1: a = b
L2:

     cmp r_b, 0
    jnz L2
L1: mov r_a, r_b
L2:

     cmp ecx, 0
cmovz [ebp+8], ecx
**Simplified Compiler Structure**

Source code (character stream)

if (b == 0) a = b;

Lexical analysis

Token stream

Parsing

Abstract syntax tree

Intermediate Code Generation

Intermediate code

Control flow graphs

Assembly Code generation

Assembly code

cmp 0, %rcx

cmovz %rcx, %rdx

**Even Bigger Picture**

Source code

Compiler

Assembly code

Assembler

Object code (machine code + symbol tables)

Fully-resolved object code (machine code + symbol tables, relocation info)

Linker

Executable image in memory

Loader
Where to Learn More

– Compilers—Principles, Techniques and Tools. Aho, Lam, Sethi and Ullman (The Dragon Book)
  (strength: parsing)

– Modern Compiler Implementation in Java. Andrew Appel.
  (strength: translation)

– Advanced Compiler Design and Implementation. Steve Muchnick.
  (strength: optimization)