
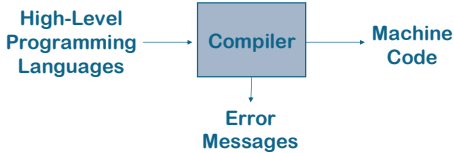


CS 471 Compilers

Prof. Kim Hazelwood
Fall 2007



What is a Compiler?



```

graph LR
    A[High-Level Programming Languages] --> B[Compiler]
    B --> C[Machine Code]
    B --> D[Error Messages]
  
```

1

CS 471 – Fall 2007

Compilers

Fundamental tool in computer science since 1952

Remains a vibrant research topic

- Machines keep changing
- Languages keep changing
- Applications keep changing
- When to compile keeps changing

Challenging!

- Must correctly handle an infinite set of legal programs
- Is itself a large program → “Where theory meets practice”

2

CS 471 – Fall 2007

Goals of a Compiler

A compiler’s job is to

- Lower the abstraction level
- Eliminate overhead from language abstractions
- Map source program onto hardware efficiently
 - Hide hardware weaknesses, utilize hardware strengths
- Equal the efficiency of a good assembly programmer


Optimizing compilers should *improve* the code

- Performance*
- Code size
- Security
- Reliability
- Power consumption

3

CS 471 – Fall 2007

An Interface to High-Level Languages



```

graph LR
    A[High-Level Programming Languages] --> B[Compiler]
    B --> C[Machine Code]
  
```

Programmers write in high-level languages

- Increases productivity
- Easier to maintain/debug
- More portable

HLLs also protect the programmer from low-level details

- Registers and caches – the `register` keyword
- Instruction selection
- Instruction-level parallelism

The catch: HLLs are less efficient

- but ... we have fast machines!

4

CS 471 – Fall 2007

High-Level Languages and Features

C (80’s) ... C++ (Early 90’s) ... Java (Late 90’s)
Each language had features that spawned new research

C/Fortran/COBOL

- User-defined aggregate data types (arrays, structures)
- Control-flow and procedures
- Prompted data-flow optimizations

C++/Simula/Modula II/Smalltalk

- Object orientation (more, smaller procedures)
- Prompted inlining

Java

- Type safety, bounds checking, garbage collection
- Prompted bounds removal, dynamic optimization

5

CS 471 – Fall 2007

An Interface to Computer Architectures

```

    graph LR
      A[High-Level Programming Languages] --> B[Compiler]
      B --> C[Machine Code]
  
```

Parallelism

- Instruction level
 - multiple operations at once
 - want to minimize dependences
- Processor level
 - multiple threads at once
 - want to minimize synchronization

Memory Hierarchies

- Speed and size are inversely proportional
- Register allocation (only portion explicitly managed in SW)
- Code and data layout (helps the hardware cache manager)

Designs driven by how well compilers can leverage new features!

6 CS 471 - Fall 2007

How Can We Translate Effectively?

```

    graph TD
      A[High-Level Source Code] --> B[?]
      B --> C[Low-Level Machine Code]
  
```

7 CS 471 - Fall 2007

Idea: Translate in Steps

- Series of program representations
- Intermediate representations optimized for program various manipulations (checking, optimization)
- More machine specific, less language specific as translation proceeds

8 CS 471 - Fall 2007

Simplified Compiler Structure

```

    graph TD
      A["Source code (character stream)  
if (b==0) a = b;"] --> B[Lexical Analysis]
      B --> C[Token stream]
      C --> D[Parsing]
      D --> E[Abstract syntax tree]
      E --> F[Intermediate Code Generation]
      F --> G[Intermediate code]
      G --> H[Code Generation]
      H --> I["Assembly code (character stream)  
CMP CX, 0  
CMOVZ CX, DX"]
      
      subgraph Front_End [Front End]
        B
        C
        D
        E
      end
      
      subgraph Back_End [Back End]
        F
        G
        H
      end
      
      subgraph Machine_Independent [Machine independent]
        B
        C
        D
        E
        F
      end
      
      subgraph Machine_Dependent [Machine dependent]
        G
        H
      end
  
```

9 CS 471 - Fall 2007

Internal Compiler Structure –Front End

Series of filter passes

```

    graph LR
      A[Source Program] --> B[Lexical Analyzer]
      B --> C[Token Stream]
      C --> D[Parser]
      D --> E[Syntax Tree]
      E --> F[Semantic Analyzer]
      F --> G[Syntax Tree]
      G --> H[Intermediate Code Gen]
      H --> I[IR]
  
```

- Source program – Written in a HLL
- Lexical analysis – Convert keywords into “tokens”
- Parser – Forms a syntax “tree” (statements, expressions, etc.)
- Semantic analysis – Type checking, etc.
- Intermediate code generator – Three-address code, interface for back end

10 CS 471 - Fall 2007

Internal Compiler Structure –Back End

```

    graph LR
      A[IR] --> B[Code Optimizer]
      B --> C[IR]
      C --> D[Code Generator]
      D --> E[Target program]
  
```

Code optimization – “improves” the intermediate code (most time is spent here)

- Consists of machine independent & dependent optimizations

Code generation

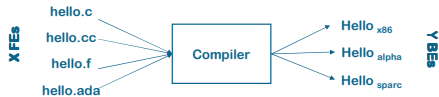
- Register allocation, instruction selection

Covered briefly at the end of the semester
Covered in detail in CS 771

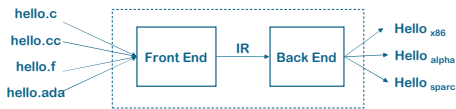
11 CS 471 - Fall 2007

Why Separate the Front and Back Ends?

Recall: An interface between HLLs and architectures



Option: X*Y compilers – or – X-front ends + Y-back ends



12

CS 471 – Fall 2007

Available Compiler Infrastructures

GNU GCC

- Targets: everything (pretty much)
- Strength: Targets everything (pretty much)
- Weakness: Not as extensible as research infrastructures, poor optimization

Stanford SUIF Compiler with Harvard MachSUIF

- Targets: Alpha, x86, IPF, C
- Strength: high level analysis, parallelization on scientific codes

Intel Open Research Compiler (ORC)

- Targets: IPF
- Strength: robust with OK code quality
- Weakness: Many IR levels

... or write your own ... (that's us)

13

CS 471 – Fall 2007

GCC Demo

14

CS 471 – Fall 2007

What will we learn in this course?

- Structure of Compilers
- Lexical Analysis
- Parsers
- Grammars, Actions, and Parse Trees
- Scoping
- Symbol Tables
- Type Checking
- Run-time Data Structures
- Intermediate Code
- Code Generation
- Advanced Topics
 - Just-in-time compilation
 - Dynamic optimization

15

CS 471 – Fall 2007

What will we learn in this course?

Theory AND practice

Other Skills

- Software engineering
- C programming
- Debugging
- Makefiles
- Clean code

16

CS 471 – Fall 2007

Course Disclaimer

Compiler courses are famously intense

- You will work hard
- You will learn a lot
- Could be the largest program you develop as an undergrad

The Workload

- You will write an actual compiler (for a toy PL)
- Subprojects build upon each other (you CANNOT fall behind)

The Good News

- Very impressive on the résumé!

17

CS 471 – Fall 2007

Required Work

Two Non-Cumulative Exams (20% each)

- October 3 and November 28

Compiler Implementation Assignments (60%)

- Roughly 8-10 assignments

Compiler Wars (End of semester)

Late Policy

- $2^{\text{days late}}$ points off (where days late > 0)

18

CS 471 - Fall 2007



Course Materials

We will use Appel's book

- Three versions - we're using C
- Will follow book project
- Code contains the occasional error...

Course Website

- www.cs.virginia.edu/kim/courses/cs471/

Lecture Slides

- On the course website
- I will try to post them before class
- Attending class is in your best interest



19

CS 471 - Fall 2007



First Steps

Get your environment set up

- Cygwin or unix (I will use Linux)
- Login to realitytv03.cs.virginia.edu
- Ssh and scp
- Makefiles
- Download the compiler modules from Appel's page
- Look at the errata (corrections) and edit your book!

Start on the primer assignment

- Due in one week

20

CS 471 - Fall 2007



Next Time...

Read Chapter 1

We will begin discussing lexical analysis

Please fill out the questionnaire!

21

CS 471 - Fall 2007

