Compilers

Lecture 1

Introduction

Yannis Smaragdakis, U. Athens
(original slides by Sam Guyer@Tufts)
Discussion

• What does a compiler do?

• Why do you need that?

• Name some compilers you have used
A Brief History of High-Level Languages

- 1953 IBM develops the 701
  - Memory: 4096 words of 36 bits
  - Speed: 60 msec for addition
  - All programming done in assembly code
Programming

- **What’s the problem?**
  - Assembly programming very slow and error-prone
  - Software costs exceeded hardware costs!

- **John Backus: “Speedcoding”**
  - Simulate a more convenient machine
  - But, ran 10-20 times slower than hand-written assembly

- **Backus**
  - **Idea**: translate high-level code to assembly
  - Many thought this impossible
    - *Had already failed in other projects*

- **1954-7 FORTRAN I project**
  - By 1958, >50% of all code is in FORTRAN
  - Cut development time dramatically – *from weeks to hours*
FORTRAN I

- The first compiler
  - Huge impact on computer science
  - Produced code almost as good as hand-written

- Led to an enormous body of work
  - Theoretical work on languages, compilers
  - Program semantics
  - Thousands of new languages

- Modern compilers preserve the outlines of FORTRAN I
Language implementations

- Two major strategies:
  - Interpretation
  - Compilation

Can you think of another strategy – a “hybrid”?

- What are the main differences?
  - “Online”: read program, execute immediately
  - “Offline”: convert high-level program into assembly code

- Compilation is a language translation problem
  - What are the languages?
Languages involved

```c
int i = 10;
while (i > 0) {
    x = x * 2;
    i = i - 1;
}
```

Source

```assembly
movl %esp, %ebp
subl $4, %esp
movl $10, -4(%ebp)
.L2:
cmpl $0, -4(%ebp)
jle .L3
movl 8(%ebp), %eax
sall %eax
movl %eax, 8(%ebp)
leal -4(%ebp), %eax
decl (%eax)
jmp .L2
.L3:
movl 8(%ebp), %eax
```

Target
The compilation problem

• Assembly language
  • Converts trivially into machine code
  • No abstraction: load, store, add, jump, etc.
  • Extremely painful to program
  • What are other problems with assembly programming?

• High-level language
  • Easy to understand and maintain
  • Abstractions: control (loops, branches); data (variables, records, arrays); procedures
  • **Problem**: how do we get from one to the other? *(systematically)*
Translation process

---

High-level language

- Letters
- Words
- Sentences

Meaning

Assembly/machine code

- Letters
- Words
- Sentences
Sounds easy!

- Translation can be tricky…

  *Infallible source: the Internet*

- I saw the Pope ("el Papa") → I saw the potato ("la papa")
- It won't leak in your pocket and embarrass you ("no los embarass") → It won't leak in your pocket and make you pregnant ("no embarazado")
- It takes a tough man to make a tender chicken → It takes a hard man to make a chicken affectionate
Job #1

- What is our primary concern?
  *Words or code: translate it correctly*

- How do we know the translation is correct?
  *Specifically, how do we know the resulting machine code does the same thing*

- “Does the same thing”
  *What does that even mean?*
Correctness

• **Practical solution**: automatic tools
  - Parser generators, regular expressions, rewrite systems, dataflow analysis frameworks, code generator-generators
  - Extensive testing

• **Theoretical solution**: a bunch of math
  - Formal description of semantics
  - A proof that the translation is correct

Topic of current research
Incorrectness

- What is this?
  
  *The infamous “Blue Screen of Death”*

- Internal failure in the operating system

- Buggy device driver
Good enough?

• Is there more than correctness?

Our wines leave you nothing to hope for.

-Swiss menu

When passenger of foot heave in sight, tootle the horn. Trumpet him melodiously at first, but if he still obstacles your passage then tootle him with vigor.

-Car rental brochure

Drop your pants here for best results.

-Tokyo dry cleaner
Job #2

• Produce a “good” translation

• What does that mean for compilers?
  
  *Good performance – optimization*
  
  • Reduce the amount of work (“be concise”)
  • Utilize the hardware effectively (“choose your words carefully”)

• How hard could that be?
Past processors

8086
29,000 transistors

Pentium M
140,000,000 transistors

• More speed, more complexity
• But, same machine code – why is that nice?
Tomorrow’s processors

- Intel Core Duo
  - Parallel, heterogeneous
    - Really hard to program!
- Xbox 360
- PS-3 CELL
Structure of a compiler

Front End

Letters \rightarrow Words \rightarrow Sentences \rightarrow Meaning

Back End

Words \rightarrow Sentences \rightarrow Letters
Structure of a compiler

• Organized as a series of passes
  • Lexical Analysis
  • Parsing
  • Semantic Analysis
  • Optimization
  • Code Generation

• We will follow this outline in the class
What I want you to get out of this class

- Understand how compilers work
  - Duh

- See how theory and practice work together
  - Yes, theory of computation is good for something
  - Also: graph algorithms, lattice theory, more...

- Work with a large-ish software systems

- Learn to think about tradeoffs
  - System design always involves tradeoffs
  - Impossible to maximize everything
Study of compilers

- Brings together many parts of CS
  - Practical and theoretical
  - Some solved problems, others unsolved
Course Structure

Course has theoretical and practical aspects

- Programming assignments = practice
  - Three homework projects
  - 55% of final grade

- Final exam: 50%
- Need to pass both exam and projects

Late policy:
Up to five late days per assignment, 5% penalty per day
Project

- Build a compiler for a subset of Java
  - Implemented in Java