Principles of Compiler Design

Amey Karkare
Department of Computer Science and Engineering, IIT Kanpur
karkare@iitk.ac.in
http://www.cse.iitk.ac.in/~karkare/cs335/
Acknowledgements

• Most of the text in the slide is based on classic text *Compilers: Principles, Techniques, and Tools* by Aho, Sethi, Ullman and Lam

• Slides are modified version of those created by Prof S K Aggarwal, IITK
Motivation

• Language processing is an important component of programming
Motivation

• Language processing is an important component of programming

• A large number of systems software and application programs require structured input
  - Operating Systems (command line processing)
  - Databases (Query language processing)
  - Type setting systems like Latex
Motivation

• Language processing is an important component of programming

• A large number of systems software and application programs require structured input
  - Operating Systems (command line processing)
  - Databases (Query language processing)
  - Type setting systems like Latex

• Software quality assurance and software testing
Motivation

• Where ever input has a structure one can think of language processing
Motivation

• Where ever input has a structure one can think of language processing

• Why study compilers?
  - Compilers use the whole spectrum of language processing technology
Expectations?

• What will we learn in the course?
What do we expect to achieve by the end of the course?

- Knowledge to design, develop, understand, modify/enhance, and maintain compilers for (even complex!) programming languages
What do we expect to achieve by the end of the course?

• Knowledge to design, develop, understand, modify/enhance, and maintain compilers for (even complex!) programming languages

• Confidence to use language processing technology for software development
Organization of the course

- Assignments 10%
- Mid semester exam 20%
- End semester exam 35%
- Course Project 35%
  - Group of 2/3/4 (to be decided)
- Tentative
Bit of History

- How are programming languages implemented? Two major strategies:
  - Interpreters (old and much less studied)
  - Compilers (very well understood with mathematical foundations)
Bit of History

• How are programming languages implemented? Two major strategies:
  - Interpreters (old and much less studied)
  - Compilers (very well understood with mathematical foundations)

• Some environments provide both interpreter and compiler. Lisp, scheme etc. provide
  - Interpreter for development
  - Compiler for deployment
Bit of History

- How are programming languages implemented? Two major strategies:
  - Interpreters (old and much less studied)
  - Compilers (very well understood with mathematical foundations)

- Some environments provide both interpreter and compiler. Lisp, scheme etc. provide
  - Interpreter for development
  - Compiler for deployment

- Java
  - Java compiler: Java to interpretable bytecode
  - Java JIT: bytecode to executable image
Some early machines and implementations

• IBM developed 704 in 1954. All programming was done in assembly language. Cost of software development far exceeded cost of hardware. Low productivity.
Some early machines and implementations

• IBM developed 704 in 1954. All programming was done in assembly language. Cost of software development far exceeded cost of hardware. Low productivity.

• Speedcoding interpreter: programs ran about 10 times slower than handwritten assembly code
Some early machines and implementations

• IBM developed 704 in 1954. All programming was done in assembly language. Cost of software development far exceeded cost of hardware. Low productivity.

• Speedcoding interpreter: programs ran about 10 times slower than hand written assembly code

• John Backus (in 1954): Proposed a program that translated high level expressions into native machine code. Skeptism all around. Most people thought it was impossible
Some early machines and implementations

- IBM developed 704 in 1954. All programming was done in assembly language. Cost of software development far exceeded cost of hardware. Low productivity.

- Speedcoding interpreter: programs ran about 10 times slower than hand written assembly code

- John Backus (in 1954): Proposed a program that translated high level expressions into native machine code. Skepticism all around. Most people thought it was impossible

- Fortran I project (1954-1957): The first compiler was released
Fortran I

- The first compiler had a huge impact on the programming languages and computer science. The whole new field of compiler design was started
Fortran I

• The first compiler had a huge impact on the programming languages and computer science. The whole new field of compiler design was started

• More than half the programmers were using Fortran by 1958
Fortran I

- The first compiler had a huge impact on the programming languages and computer science. The whole new field of compiler design was started

- More than half the programmers were using Fortran by 1958

- The development time was cut down to half
Fortran I

• The first compiler had a huge impact on the programming languages and computer science. The whole new field of compiler design was started

• More than half the programmers were using Fortran by 1958

• The development time was cut down to half

• Led to enormous amount of theoretical work (lexical analysis, parsing, optimization, structured programming, code generation, error recovery etc.)
Fortran I

- The first compiler had a huge impact on the programming languages and computer science. The whole new field of compiler design was started

- More than half the programmers were using Fortran by 1958

- The development time was cut down to half

- Led to enormous amount of theoretical work (lexical analysis, parsing, optimization, structured programming, code generation, error recovery etc.)

- Modern compilers preserve the basic structure of the Fortran I compiler !!!
The big picture

• Compiler is part of program development environment

• The other typical components of this environment are editor, assembler, linker, loader, debugger, profiler etc.

• The compiler (and all other tools) must support each other for easy program development
Programmer → Editor → Compiler → Assembler

Source Program → Assembly code → Machine Code
Programmer → **Editor** → **Compiler** → **Assembler** → **Linker** → **Loader** → Execution on the target machine

- Source Program
- Assembly code
- Machine Code
- Resolved Machine Code
- Executable Image
Programmer → **Editor** → **Compiler** → **Assembler** → **Linker** → **Loader** → **Execution on the target machine**

Normally end up with error
Programmer → **Editor** → **Compiler** → **Assembler** → **Linker** → **Loader** → **Debugger**

- **Source Program** → **Assembly code**
- **Machine Code**
- **Resolved Machine Code**
- **Executable Image**
- **Debugging results**
- **Execute under Control of debugger**
- **Execution on the target machine**

Normally end up with error
Programmer does manual correction of the code

Debugging results

Execute under Control of debugger

Execution on the target machine

Normally end up with error