

# *Overview of the Course*

Copyright 2003, Keith D. Cooper, Ken Kennedy & Linda Torczon, all rights reserved.  
Students enrolled in Comp 412 at Rice University have explicit permission to make copies of these materials for their personal use.

## Class-taking technique for Cmp Sci 410/610

---

- I will use projected material sometimes (trying it out)
  - I will moderate my speed, *you* sometimes need to say "STOP"
- You should read the book
  - Not all material will be covered in class
  - Book complements the lectures
- You are responsible for material from class
  - The tests will cover both lecture and reading
- Cmp Sci 410/610 is not a *programming* course
  - Projects are graded on functionality more than style  
(*results matter*)
- It will take me time to learn your names (please remind me)

# Compilers

---

- What is a **compiler**?

# Compilers

---

- What is a **compiler**?
  - A program that translates an *executable* program in one language into an *executable* program in another language
  - The compiler should improve the program, *in some way*
- What is an **interpreter**?

# Compilers

---

- What is a **compiler**?
  - A program that translates an *executable* program in one language into an *executable* program in another language
  - The compiler should improve the program, *in some way*
- What is an **interpreter**?
  - A program that reads an *executable* program and produces the results of executing that program

# Compilers

---

- What is a **compiler**?
  - A program that translates an *executable* program in one language into an *executable* program in another language
  - The compiler should improve the program, *in some way*
- What is an **interpreter**?
  - A program that reads an *executable* program and produces the results of executing that program
- C is typically compiled, Scheme is typically interpreted
- Java is compiled to bytecodes (code for the Java VM)
  - which are then interpreted
  - Or a hybrid strategy is used
    - Just-in-time compilation

# Taking a Broader View

---

- Compiler Technology = Off-Line Processing
  - **Goals:** improved performance and language usability
    - Making it practical to use the full power of the language
  - **Trade-off:** preprocessing time versus execution time (or space)
  - **Rule:** performance of both compiler and application must be acceptable to the end user
- Examples
  - Macro expansion
    - PL/I macro facility — 10x improvement with compilation

# Taking a Broader View

---

- Compiler Technology = Off-Line Processing
  - **Goals:** improved performance and language usability
    - Making it practical to use the full power of the language
  - **Trade-off:** preprocessing time versus execution time (or space)
  - **Rule:** performance of both compiler and application must be acceptable to the end user
- Examples
  - Macro expansion
    - PL/I macro facility — 10x improvement with compilation
  - Database query optimization

# Taking a Broader View

---

- Compiler Technology = Off-Line Processing
  - **Goals:** improved performance and language usability
    - Making it practical to use the full power of the language
  - **Trade-off:** preprocessing time versus execution time (or space)
  - **Rule:** performance of both compiler and application must be acceptable to the end user
- Examples
  - Macro expansion
    - PL/I macro facility — 10x improvement with compilation
  - Database query optimization
  - Emulation acceleration
    - TransMeta "code morphing"

# Why Study Compilation?

---

- Compilers are important system software components
  - They are intimately interconnected with architecture, systems, programming methodology, and language design
- Compilers include many applications of theory to practice
  - Scanning, parsing, static analysis, instruction selection
- Many practical applications have embedded languages
  - Commands, macros, formatting tags ...
- Many applications have input formats that look like languages,
  - Matlab, Mathematica
- Writing a compiler exposes practical algorithmic & engineering issues
  - Approximating hard problems; efficiency & scalability

# Intrinsic interest

---

- Compiler construction involves ideas from many different parts of computer science

<i>Artificial intelligence</i>	Greedy algorithms Heuristic search techniques
<i>Algorithms</i>	Graph algorithms, union-find Dynamic programming
<i>Theory</i>	DFAs & PDAs, pattern matching Fixed-point algorithms
<i>Systems</i>	Allocation & naming, Synchronization, locality
<i>Architecture</i>	Pipeline & hierarchy management Instruction set use

# Intrinsic merit

---

- Compiler construction poses challenging and interesting problems:
  - Compilers must do a lot but also **run fast**
  - Compilers have primary responsibility for **run-time performance**
  - Compilers are responsible for making it acceptable to use the **full power** of the programming language
  - Computer architects perpetually create new challenges for the compiler by building more **complex machines**
  - Compilers must hide that complexity from the programmer
  - Success requires mastery of complex interactions

## Making Languages Usable

---

It was our belief that if FORTRAN, during its first months, were to translate any reasonable “scientific” source program into an object program only half as fast as its hand-coded counterpart, then acceptance of our system would be in serious danger... I believe that had we failed to produce efficient programs, the widespread use of languages like FORTRAN would have been seriously delayed.

— John Backus

# About the instructor

---

- My own research
  - Compiling advanced features for advanced architectures
  - Garbage collection (automatic storage management)
  - Efficient simulation (think JIT compilation)
  - Automatic generation of compiler & simulator components
- Thus, my interests lie in
  - Quality of generated code
  - Interplay between compiler and architecture
  - Run-time performance analysis
  - Dynamic performance improvement
  - Automatic processing of machine descriptions into tools

## Next set of slides

---

- The view from 35,000 feet
  - How a compiler works
  - What I (i.e., the author) think is important
  - What is hard and what is easy