Welcome to UG3 **Compiling Techniques**

**Topics in the design of programming language translators, including parsing, run-time storage management, error recovery, code generation, and optimization**

- **Instructor:** Dr. Björn Franke (bfranke@inf.ed.ac.uk)
- **Office Hours:** Monday 2 PM to 3 PM, JCMB 2414
- **Text:** Keith Cooper & Linda Torczon - Engineering a Compiler
  - Textbook can be reused in **UG4 Compiler Optimisation** course
- **Web Site:** [http://www.inf.ed.ac.uk/teaching/courses/ct/](http://www.inf.ed.ac.uk/teaching/courses/ct/)
  - Coursework, slides (2 per page), practice exams, ...
  - I will not have handouts in class; get them from the web
- **Slides:** Closely based on Keith Cooper’s slides
  - Selection of approx. 15 out of >35 lectures
  - Dropped optimisation, smaller amount of in-depth material
Basis for Grading

- **Exams**
  - Final 75%

- **Coursework**
  - Lexer & Parser 12.5%
  - Dataflow Analysis 12.5%
<table>
<thead>
<tr>
<th>Topic</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>§ 1</td>
</tr>
<tr>
<td>Local Register Allocation</td>
<td>§ 13.2</td>
</tr>
<tr>
<td>Scanning</td>
<td>§ 2</td>
</tr>
<tr>
<td>Parsing</td>
<td>§ 3</td>
</tr>
<tr>
<td>Context Sensitive Analysis</td>
<td>§ 4</td>
</tr>
<tr>
<td>Inner Workings of Compiled Code</td>
<td>§ 6, 7</td>
</tr>
<tr>
<td>Introduction to Optimization</td>
<td>§ 8</td>
</tr>
<tr>
<td>Code Selection</td>
<td>§ 11</td>
</tr>
<tr>
<td>Instruction Scheduling</td>
<td>§ 12</td>
</tr>
<tr>
<td>Register Allocation</td>
<td>§ 13</td>
</tr>
<tr>
<td>More Optimization (time permitting)</td>
<td></td>
</tr>
</tbody>
</table>

If it looks like the course follows the text, that's because the text was written from the course.

What about the missing chapters?

5: We'll fit it in

9, 10: see UG4 Compiler Optimisation
Class-taking technique for Compiling Techniques

- I will use projected material extensively
  - 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 exam will cover both lecture and reading
  - I will probably hint at good test questions in class

- “Compiling Techniques” is not a programming course
  - Coursework is graded on functionality and documentation more than style (*results matter*)
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

| Artificial intelligence | Greedy algorithms  
|-------------------------|----------------------
|                         | Heuristic search techniques |
| Algorithms              | Graph algorithms, union-find  
|                         | Dynamic programming |
| Theory                  | DFAs & PDAs, pattern matching  
|                         | Fixed-point algorithms |
| Systems                 | Allocation & naming,  
|                         | Synchronization, locality |
| Architecture            | 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**
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 for embedded processors
    ▪ Optimisation for embedded systems (space, power, speed)
      - Source-level transformation
      - Adaptive compilation
    ▪ Parallelisation for multi-core embedded systems
      - Homogeneous targets, e.g. Multi-DSP
      - Heterogeneous targets, e.g. Systems-on-Chip
  → Design Space Exploration
    ▪ Architecture & Compiler Synthesis

• Thus, my interests lie in
  → Quality of generated code
  → Interplay between application, compiler and architecture
Next class

- The view from 35,000 feet
  - How a compiler works
  - What I think is important
  - What is hard and what is easy