The University for world-class professionals

Manchester Metropolitan University

Compilers: Overview

Dr Paris Yiapanis
room: John Dalton E151
email: p.yiapanis@mmu.ac.uk

Today

- Contact details
- Administrative details
- References and reading materials
- Outline of material
- Why study compilers
- Overview of compilers

About me

- Dr Paraskevas (Paris) Yiapanis
- email: p.yiapanis@mmu.ac.uk
- room: JD E151
- office hours: which 3 hours?

  (most up-to-date hours are always in my Moodle profile)

Lectures

- I will try to make available lecture slides on Moodle each week, by the Friday prior to the lecture.
  - I will not provide printouts.
  - I tend to say a lot more than is written on my slides.
    - **Skipping lectures is a bad idea**

Labs

- Labs this term are **completely** different to last term.
  - Lab work will be assessed on the exam
  - You will **only** get attendance for the lab scheduled in your timetable.
  - There are more students per lab this term so attend only the lab scheduled in your timetable to ensure a machine available and sufficient tutor time per student

Time

- One hour-long lecture per week
  - Mostly theory, assessed on the exam
- One two-hour-long lab class per week
  - Working on your coursework assessment.
- 30 credits for unit → 15 credits this term
  - 15 credits ≈ 150 hours of work
    - 150 – (1+2)*12 = 114
    - 114/12 = 9.5 self-study hours per week!

Assessment (and Time Management)

- Assessment is only exam this term
  - But will assess both lab project and lectures
- This term's exam is worth 50% of your mark for this subject.
  - Lab work is staged, adding new steps each week.
  - You are **STRONGLY** encouraged to follow the suggested schedule to complete your lab project, and take advantage of the time during labs.
- The exam will be only on this term's material
  - This is different from other years
  - You will be given guidance on what to expect later in the term

Communication

- Please let me know ASAP if there are any issues that I can deal with directly
  - At the end of class (if it is brief)
  - By appointment during office hours
  - By email – if brief (please include 6G6Z1110 in the subject line)
- If you have exceptional factors/mitigating circumstances affecting your performance, submit the appropriate form through the Student Hub (I cannot deal with these directly)
- Please make sure you check your student email and Moodle regularly for subject-related announcements

Reading Material

- Recommended text:
  - *Engineering a Compiler* , 2nd edition, 2012. Cooper & Torczon (available online through MMU library)
- Alternative:
  - *Compilers: Principles, Techniques, & Tools*, 2nd edition, 2006 Aho, Lam, Sethi & Ullman.

Lectures this Term

- **Admin and overview**
- Lexical analysis
- Parsing
- Semantic analysis
- Machine-independent optimisation
- Code generation
- Hardware architectures
- Machine-dependent optimisation
- Review

Overview of Compilers

- Objectives:
  - To describe the purpose of a compiler and an interpreter
  - To justify *why* you should be studying compilers
  - To identify the key components of a compiler

What is a Compiler?

- (This *should* be recap for you!)
- Putting it in very basic terms:
  - A computer operates on electricity; essentially everything that happens is in terms of whether multitudes of tiny internal switches are off or on.
  - A compiler takes a high level description of a solution to a problem and transforms it into the 0s and 1s that the computer can use.
- In more technical terms:
  - A compiler translates an executable program in one language into an executable program in another language.

What is an Interpreter?

- An interpreter reads an executable program and produces the results of executing that program.

Why Study Compilers?

Very few people *write* compilers in "the real world"

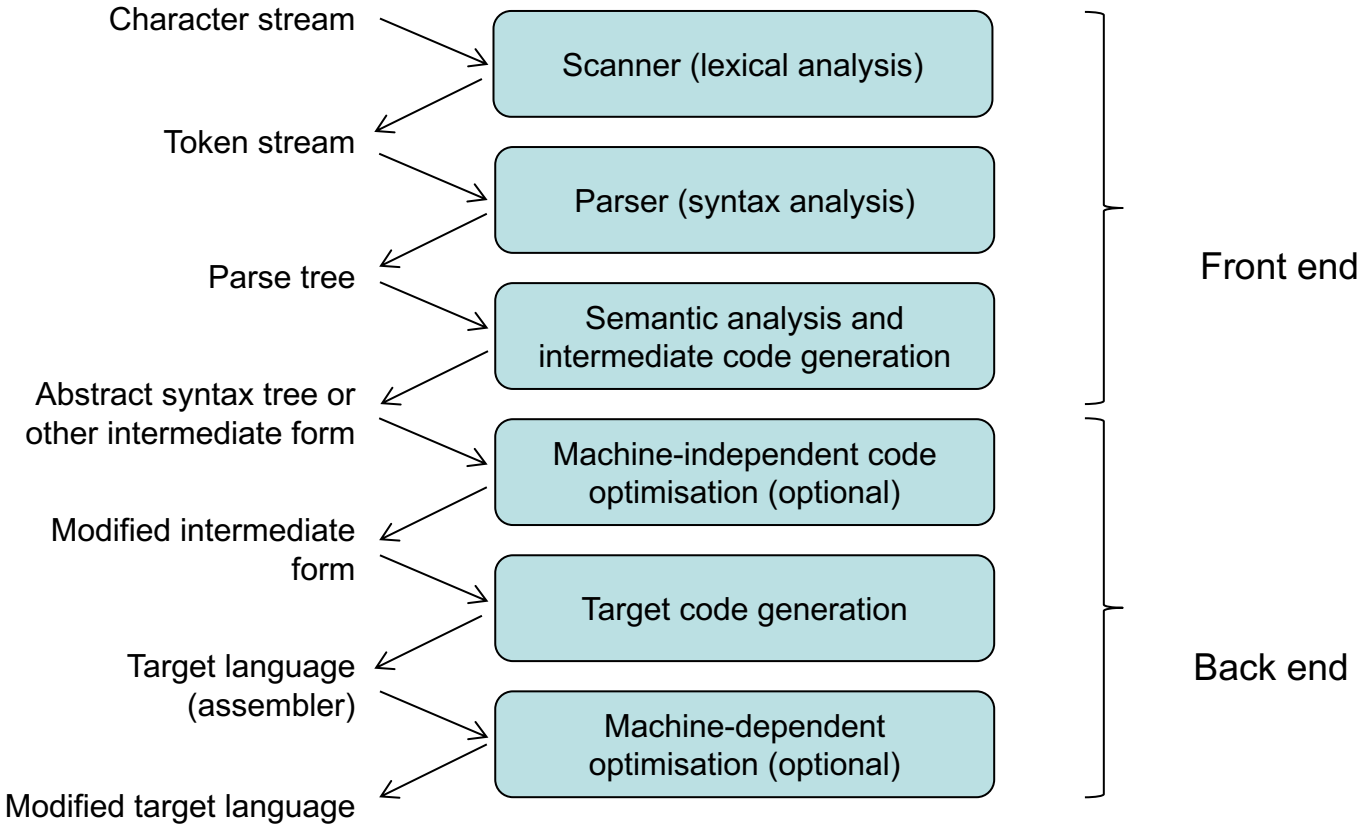
…unless they are developing a new language (for high level descriptions)

But

Compiler writing

- Gives experience in large-scale system development
- Uses techniques and algorithms that are widely used in other programming applications
- Exemplifies many theoretical aspects of Computer Science

Phases of Compilation



Scanning (Lexical Analysis)

- Input is a stream of input characters.
- Produces an output stream of words.
- Each word has a syntactic category: identifier, operator, keyword, constant, literal ...
- Word + syntactic category = a token.
- Valid words are specified as patterns of regular expressions.
- Usually semi-automated with tools like lex or ANTLR.
- Lexical analysis relies on deterministic finite automata.

Parsing (syntax analysis)

- Checks the stream of tokens produced by the scanner for grammatical correctness.
- A parser must determine if the program to be compiled is a valid sentence in the “syntactic model of the programming language”.
- Output is a concrete model of a program, ie an *intermediate representation* (IR), OR errors are raised.
- IR is used by later compilation phases.
- Syntax (grammar) is often expressed using a context free grammar.

Semantic Analysis

- Syntax ignores meaning, it checks only grammar.
- SA checks if a sentence has a valid and correct meaning.
- w ← w*2*x*y*z
- Above might be ill-formed if, one or more of the variable names is undefined/declared.
- The types associated with the variables might not support the * operator. ~~String*integer~~
- SA must check for correctness and meaning, it helps in the construction of an IR.

Examples of Semantic Analysis

- Every identifier is declared before used.
- All identifiers are used in appropriate contexts.
- Subroutine/function/method/procedure calls have the correct number and type of arguments.
- Labels on the case branches of a switch statement are distinct constants.
- Any non-void function must return a value of the correct type explicitly.

Intermediate Code Generation

- An intermediate representation (IR) encodes the compiler's knowledge about a program.
- IR decisions affect speed and efficiency of the compiler.
- Three major categories:
  - Structural: trees, graphs.
  - Linear: pseudo code for an abstract machine (stack machine code, 3 address codes).
  - Hybrid: combination of linear and structural.

Intermediate Representation Issues

- Ease of generation/manipulation:
  - computation time/complexity, one or multiple passes of parsed program and volume of data storage required.
- Freedom of expression:
  - can it record all the useful facts from previous compiler phases?
- Level of abstraction:
  - high-level or lower-level closely related to machine/assembler code.

Machine-Independent Optimisation

- Must preserve program semantics!
- Eliminate useless and unreachable code
- Code motion
  - (move code so it executes less frequently & produces same answer)
- Specialisation
  - (constant propagation ...)
- Eliminate redundancy
  - (that is, prevent repeated/unnecessary computation)
- Enable other transformations for later phases

Target Code Generation

- Implies traversal(s) of the IR.
- Emit equivalent code for the target machine (bytecode, assembler).
- Target machine instructions must be selected to match IR operations.
- Instructions must be scheduled.
- IR values/variables must be allocated to registers/memory.
- TASKS: instruction selection, register/memory allocation and instruction scheduling

Machine-Dependent Optimisation

- Exploit special purpose features/instructions.
- Manage or hide latency of operations to prevent processor stalls and keep pipelines of functional units occupied.
- Manage finite processor resources.
- Manage limited functional units and registers, avoid/prevent processor stalls.
- Many compilers offer a generic way of generating target specific code, and use a subsequent specific pass optimised for a given backend target.

Recap

- A compiler is…
- An interpreter is…
- Why study them?
- What are the key phases of compilation?

Where we are…

- ~~Admin and overview~~
- Lexical analysis
- Parsing
- Semantic analysis
- Machine-independent optimisation
- Code generation
- Hardware architectures
- Machine-dependent optimisation
- Review