© Manuel Cargaleiro

Compilers

Masters in Informatics and Computing Engineering (MIEIC), 3rd Year

João M. P. Cardoso
Dep. de Engenharia Informática, Faculdade de Engenharia (FEUP), Universidade do Porto, Porto, Portugal
Email: jmpc@fe.up.pt

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Faculty Members

- 2019/2020
  - João M. P. Cardoso
    - Office I137 OR DEI secretary's office
    - E-mail: jmpc@fe.up.pt
  - João Bispo
    - Room: J204
  - Tiago Carvalho
    - Room: J204
  - Lázaro Costa
    - Room: J204

Course Webpages

- SiFEUP (rules, timetable, list of students, etc.):
  - https://sigarra.up.pt/feup/pt/ucurr_geral.ficha_uc_view?pv_ocorrencia_id=436448
- Moodle (organization of the course, mailing-lists, etc.):
  - https://moodle.up.pt/course/view.php?id=2338
- Google Drive (files, documents, etc.)

Objectives

- Provide concepts which allow to:
  - understand the programming languages' compilation phases, in particular for imperative and object-oriented (OO) languages;
  - specify the syntax and semantics of a programming language;
  - understand and use the data structures and the main algorithms used to implement compilers;
  - build a compiler or software needing compiler topics;
  - help understanding the options of existent compilers

Learning Outcomes and Competences

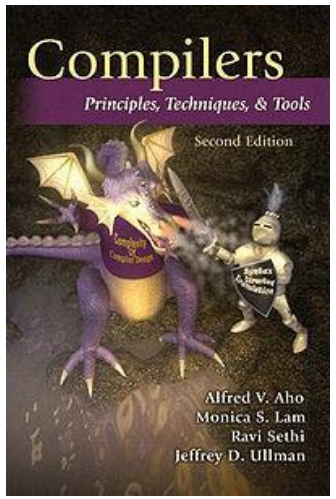
- The skills and learning outcomes will allow students to:
  - develop and implement software processing systems of artificial languages and information textually specified under certain lexical and grammar rules;
  - design and implement in software the various compiler stages, namely:
    - lexical analysis (regular expressions and finite automata)
    - syntactic analysis (context-free grammars and PDAs)
    - semantic analysis
    - code optimization
    - code generation having microprocessors or virtual machines as target

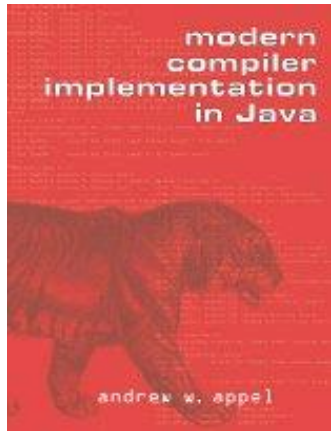
Prior Knowledge

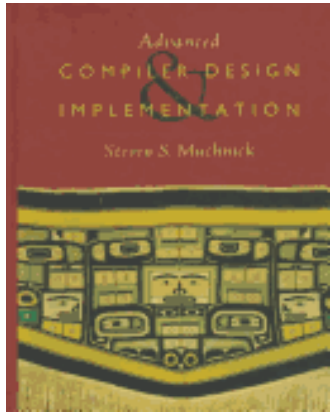
- Pre-requirements (prior knowledge) and co-requirements (common knowledge)
  - Computer Architecture
  - Imperative programming languages, object-oriented programming languages
  - Data structures and algorithms
  - Theory of Computation

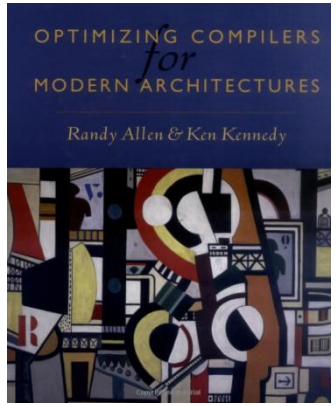
Syllabus

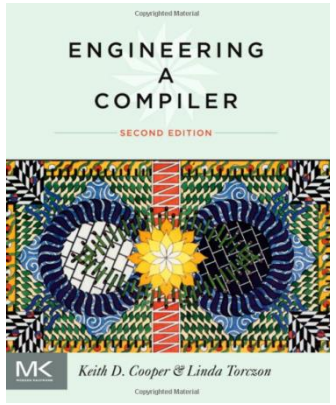
1. Introduction. Compilation phases and typical structure of a compiler.
2. Lexical analysis. Regular expressions and finite automaton.
3. Syntax analysis. Grammars. Syntax analysis' algorithms. Error handling.
4. Semantic analysis. Type checking.
5. Execution environments. Memory organization and schemes for parameter passing.
6. High and Low-level intermediate representations. Intermediate code generation techniques.
7. Code generation techniques. Instruction selection, register allocation, and scheduling.
8. Compiler optimizations.

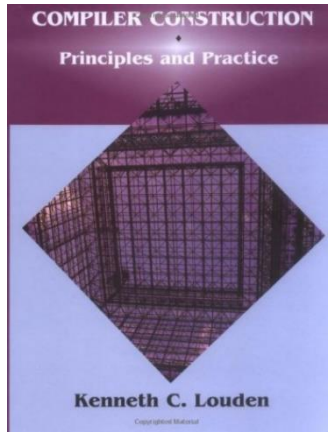












Bibliography

- **Principal**
  - A. Aho, M. Lam, R. Sethi, J. Ullman, *Compilers: Principles, Techniques, and Tools*, 2nd Edition, Addison Wesley, 2007. ISBN: 0321486811 (Existe 1ª edição (1986) na biblioteca)
  - Appel, Andrew Wilson, *Modern Compiler Implementation in Java*, 2nd edition. Cambridge University Press, 2002. ISBN 0-521-82060-X
- **Complementary**
  - Muchnick, Steven, *Advanced Compiler Design and Implementation*, Morgan Kaufman Publishers, 1997. ISBN 1-55860-320-4
  - Allen, Randy; and Kennedy, Ken, *Optimizing Compilers for Modern Architectures*, Morgan Kaufman Publishers, 2001. ISBN 1-55860-286-0
  - Cooper, Keith D., and Torczon, Linda, *Engineering a Compiler*, Morgan Kaufmann, 2nd edition, February 21, 2011. ISBN 10: 012088478X
  - Louden, Kenneth C.;Compiler construction. Course Technology, ISBN 0-534-93972-4
  - Pedro Reis Santos, Thinault Langlois; *Compiladores - da Teoria à Prática*, FCA, 2014. ISBN: 978-972-722-768-6 **[in Portuguese]**

Teaching Methods

- 2×1.5-hour classes (Ts: lectures):
  - Presentation of the topics, exercises related to compiler theory and practice
  - Discussions of ideas, solutions, etc.
- 1-hour classes (TPs):
  - Resolution and discussion of topics related to the project
  - Meeting with instructors

Lectures

- Lectures
  - Periods of 20 min. with discussion (including practical issues) and activity breaks (10 min. each)
  - Use of videos and other material to promote discussions

Assessment

- **Assessment Method:** Distributed evaluation without final exam
- **First requirement to succeed (R1): AD** (with grade >= 10) and at most 3 absences from the TP classes
  - **AD**: grade obtained in the distributed evaluation (see next slide) [0..20]
- **Final Grade**
  - **FIRST ROUND** ("Época Normal"):
    - Final Grade = ROUND(0.60*AD + 0.2*T1 + 0.2*T2)
    - T1: grade obtained in the first midterm exam [0..20]
    - T2: grade obtained in the second midterm exam [0..20]
    - Each student will succeed in the "época normal" if he/she attained the conditions for admission to exams (**R1**), obtained a minimum score in each individual midterm exams (T1 and T2) of **7 marks**, and an average grade of the tests (T1 and T2) **>= 8 marks**, and obtained a **Final Grade >= 10 marks**
  - **SECOND ROUND** ("Época de recurso"):
    - Final Grade = ROUND(0.60*AD + 0.40*EX)
    - EX: grade obtained in the exam [0..20]
    - Each student will succeed in the course in the "época de recurso" if he/she attained the conditions for admission to exams (**R1**), obtained a **minimum grade of 8 marks on the Exam (EX)**, and obtained a **Final Grade >= 10**.
  - **FIRST ROUND** and **SECOND ROUND**:
    - In order to obtain higher grades than 17 (out of 20), it is necessary to do an oral exam or additional work.

Assessment (cont.)

- **AD**: grade of the distributed evaluation (incl. project) [0..20]
- AD grade consists of:
  - Participation: 10% (including participation in lectures and regular activities via moodle)
  - First checkpoint: 5%
  - Second checkpoint: 10%
  - Third checkpoint: 10%
  - Final work: 45%
  - Presentation/Discussion: 20%

Software

- Parser generators:
  - JavaCC, https://javacc.org/
  - ANTLR - Another Tool for Language Recognition, http://www.antlr.org/
- Assemblers:
  - Jasmin – JVM assembler, http://jasmin.sourceforge.net/
- APIs and tools:
  - Graphviz - Graph Visualization Software, http://www.graphviz.org/
  - Graph libraries: JGraphT (http://jgrapht.org/), GraphStream (http://graphstream-project.org/download/)
- Compilers and parsers:
  - Clang: a C language family frontend for LLVM, http://clang.llvm.org/
  - JavaScript parser and Syntax Tree generator built in JavaScript: http://esprima.org/demo/parse.html
- Helpful environments:
  - Compiler Explorer: https://gcc.godbolt.org/
- IDEs: Eclipse, NetBeans,…

Beginning of High-Level Languages

- **Ada Lovelace** (1815-1852)
  - "The **Ada Lovelace Award** is named in honor of the first computer programmer, Augusta Ada Byron Lovelace, whose writings developed the idea of programming and explained the operation and theory of Charles Babbage's Analytical Engine."

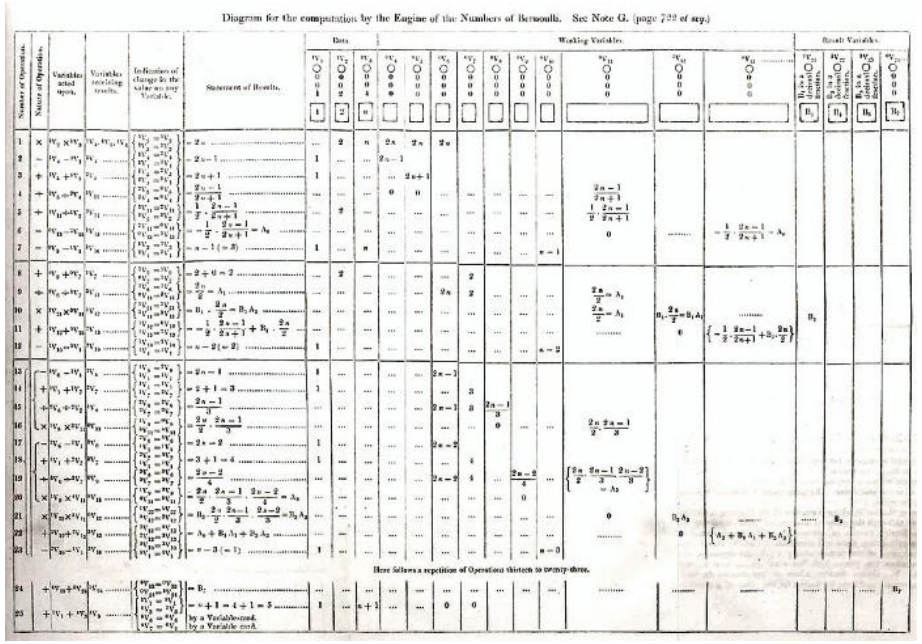
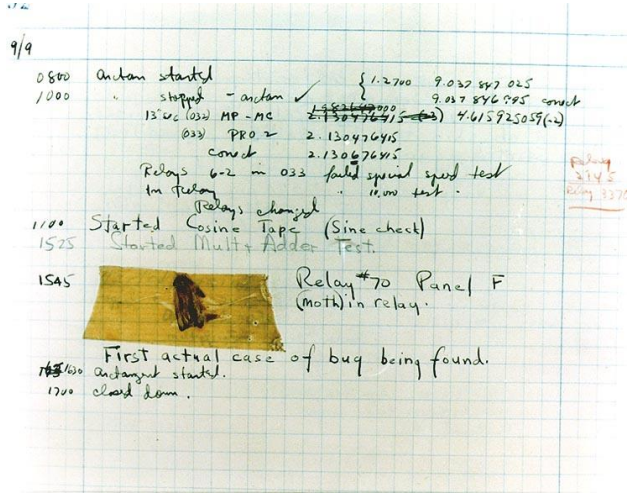


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 et seq.)

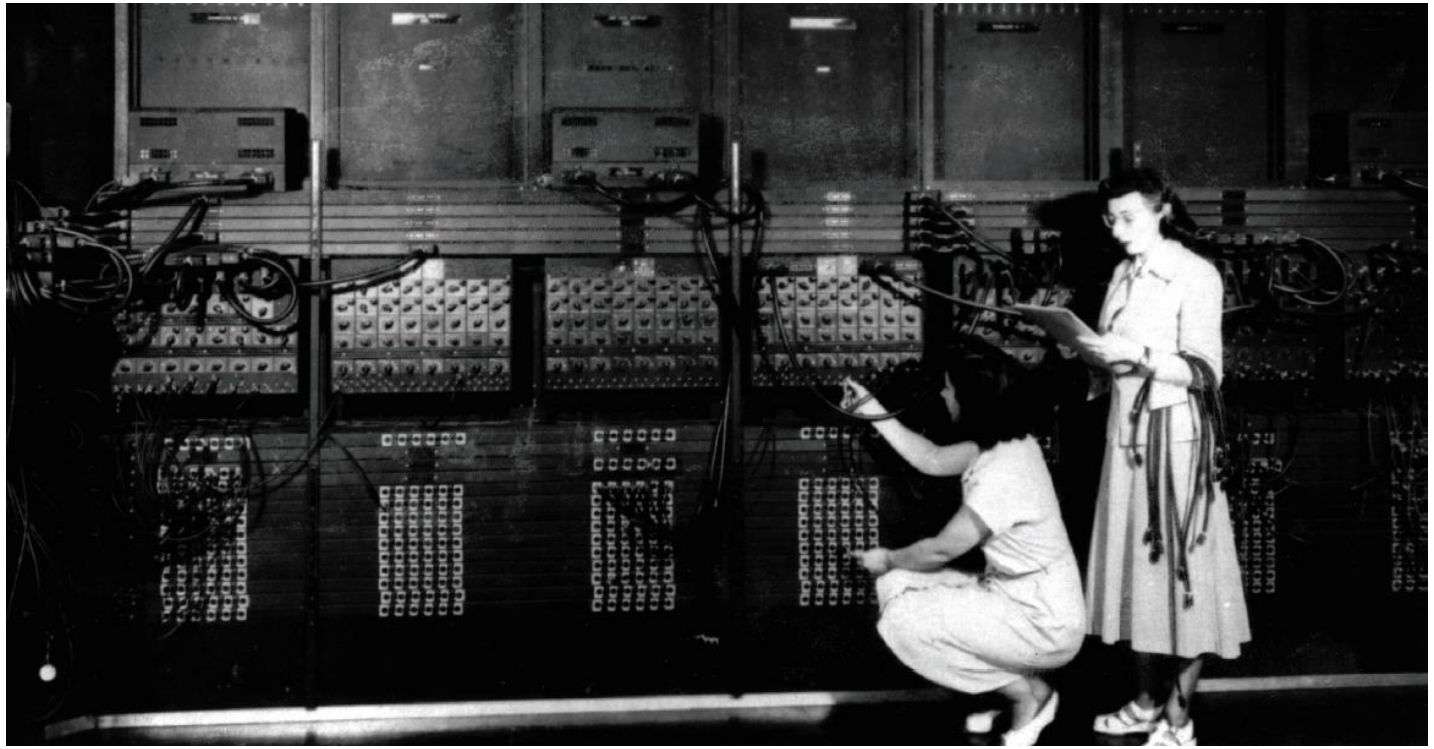
In note G, she describes an algorithm for the Analytical Engine to compute Bernoulli numbers.

Beginning of the Compiler

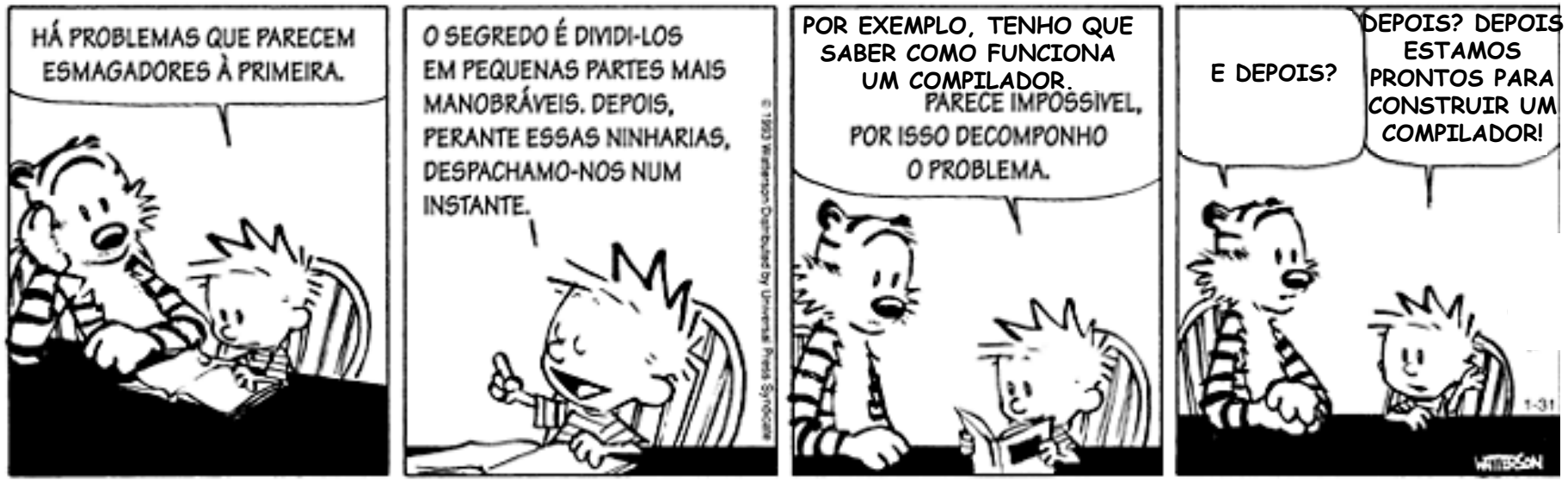
- **Grace Brewster Murray Hopper** (1906 –1992)
  - "One of the first programmers of the Harvard Mark I computer in 1944, invented the first compiler for a computer programming language, and was one of those who popularized the idea of machine-independent programming languages." [source: wikipedia]
  - Also associated to the term "bug":



- Two programmers wiring the right side of the ENIAC with a new program



Source: Actually, Turing Did Not Invent the Computer
By Thomas Haigh
Communications of the ACM, Vol. 57 No. 1, 2014, pp. 36-41


HÁ PROBLEMAS QUE PARECEM ESMAGADORES À PRIMEIRA.
O SEGREDO É DIVIDI-LOS EM PEQUENAS PARTES MAIS MANOBRÁVEIS. DEPOIS, PERANTE ESSAS NINHARIAS, DESPACHAMO-NOS NUM INSTANTE.
© 1993 Watterson Distributed by Universal Press Syndicate
POR EXEMPLO, TENHO QUE SABER COMO FUNCIONA UM COMPILADOR.
PARECE IMPOSSÍVEL, POR ISSO DECOMPONHO O PROBLEMA.
E DEPOIS?
DEPOIS? DEPOIS ESTAMOS PRONTOS PARA CONSTRUIR UM COMPILADOR!
1-31
WATTERSON

GOOD WORK!