© Manuel Cargaleiro

Compilers

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

João M. P. Cardoso
Email: jmpc@fe.up.pt

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Faculty Members

- 2020/2021
  - João M. P. Cardoso
    - Office I137 or DEI head's office
    - E-mail: jmpc@fe.up.pt
  - João Bispo
    - Room: J204
    - E-mail: jbispo@fe.up.pt
  - Tiago Carvalho
    - Room: J204
    - E-mail: tdrc@fe.up.pt
  - Lázaro Costa
    - Room: J204
    - E-mail: lazaro@fe.up.pt

Course Webpages

- SiFEUP (rules, timetable, list of students, etc.):
  - https://sigarra.up.pt/feup/pt/ucurr_geral.ficha_uc_view?pv_ocorrencia_id=459486
- MS Teams (organization of the course, chats, material, quizzes, etc.):
  - https://teams.microsoft.com/l/team/19%3a863496861fd04b7a869f33ed7e8cb752%40thread.tacv2/conversations?groupId=1d256bf6-e8f7-46fa-89e0-c21504d6f8e0&tenantId=b7821bc8-67cc-447b-b579-82f7854174fc

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

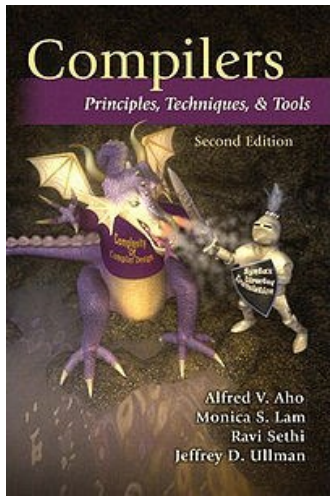
- The skills and learning outcomes will allow students to:
  - develop and implement software processing systems of artificial languages and information textually specified under certain lexical and grammar rules;
  - design and implement in software the various compiler stages, namely:
    - lexical 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

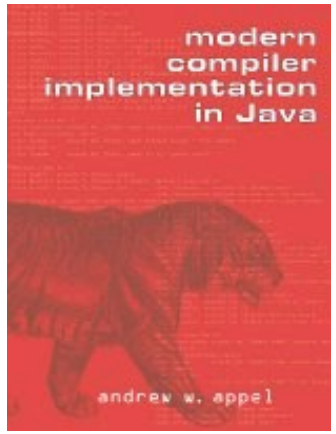
Prior Knowledge

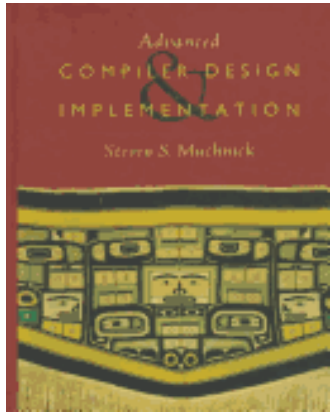
- Pre-requirements (prior knowledge) and co-requirements (common knowledge)
  - Computer Architecture
  - Imperative programming languages, object-oriented programming languages
  - Data structures and algorithms
  - Theory of Computation

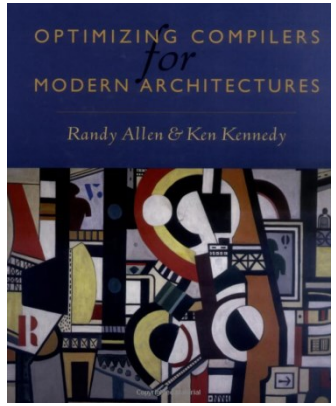
Syllabus

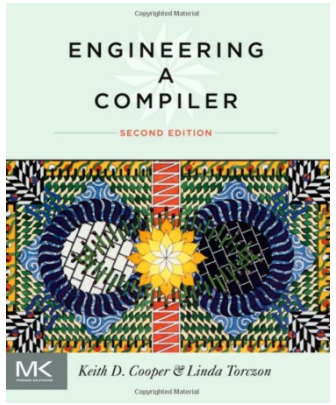
1. Introduction. Compilation phases and typical structure of a compiler.
2. Lexical analysis. Regular expressions and finite automaton.
3. Syntax analysis. Grammars. Syntax analysis' algorithms. Error handling.
4. Semantic analysis. Type checking.
5. Execution environments. Memory organization and schemes 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.

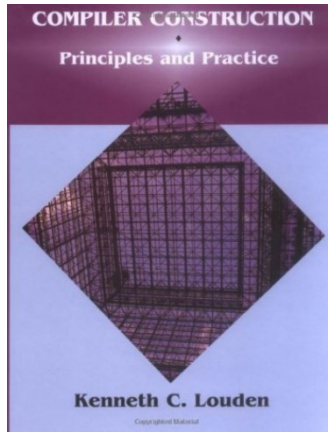












Bibliography

- **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, Feb. 21, 2011. ISBN 10: 012088478X
  - Kenneth C. Louden, Compiler Construction: Principles and Practice. Cengage Learning; 1 edition (Jan. 24, 1997), USA. 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 Rules

- **Assessment Mode**
  - Distributed evaluation with Final Exam
- **Passing in the distributed evaluation**
  - Distributed evaluation (AD) not inferior to 10.0 marks and a maximum of 3 non-justified absences (25%) on the lab (TP) classes
- **Final Grade** (all marks from 0 to 20)
  - **AD**: Distributed Evaluation consists of three components (min: 10.0 marks)
    - LHA: Lecture and homework activities (homeworks and quizzes during the lectures): 15%
    - PRJ: Compiler Project: 70%
    - CA: Challenge activities (2 challenges during the semester): 15%
    - **AD = 0.15 × LHA + 0.70 × PRJ + 0.15 × CA**
  - **FEG**: final exam grade (min: 8.0 marks), "Época Normal" (First Round Exam) or "Época de Recurso" (Second Round Exam)
  - **AD Grade (ADG)** =
    - **AD if AD** <= **FEG**+3
    - **FEG**+3, **otherwise**
  - **Final Grade** = rounded(0.70 × ADG + 0.30 × FEG)

Assessment Rules (cont.)

- **Compiler Project (PRJ)**
  - Compiler project
- **Components contributing to the PRJ grade**
  - First checkpoint: 5%
  - Second checkpoint: 10%
  - Third checkpoint: 10%
  - Final work: 55%
  - Presentation/Discussion: 20%

Assessment Rules (cont.)

- Assessment for Students under a special enrollment (TE, DA, etc.)
  - The same rules, but without the condition of a maximum of 3 non-justified absences (25%) on the lab (TP) classes
- Students who have concluded the AD with success in the previous academic year and who don't want to repeat the AD will have the final grade given by:
  - **Final Grade** = rounded(0.6 × ADG + 0.4 × FEG), where the ADG is the AD grade obtained in the previous academic year
- *Possibility to improve the exam grade (FEG) by doing a scientific work*

SOFTWARE

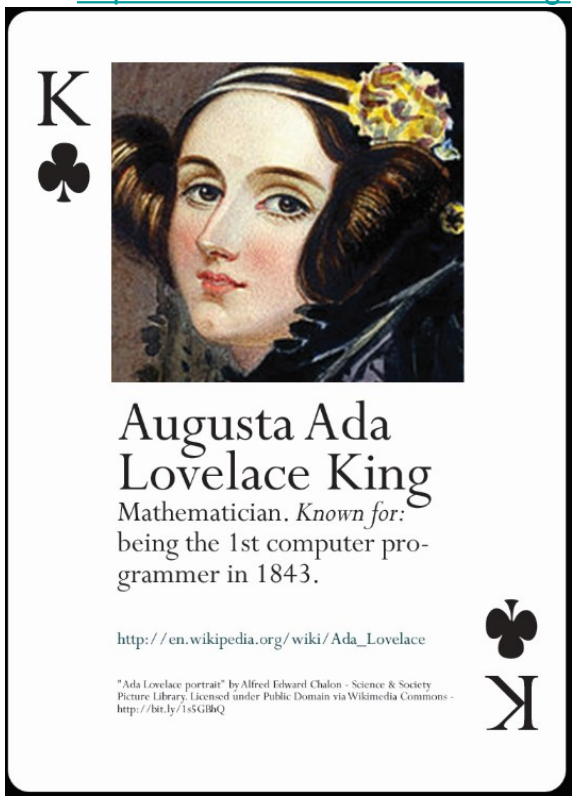
Software

- 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, Intellij IDEA, NetBeans,…

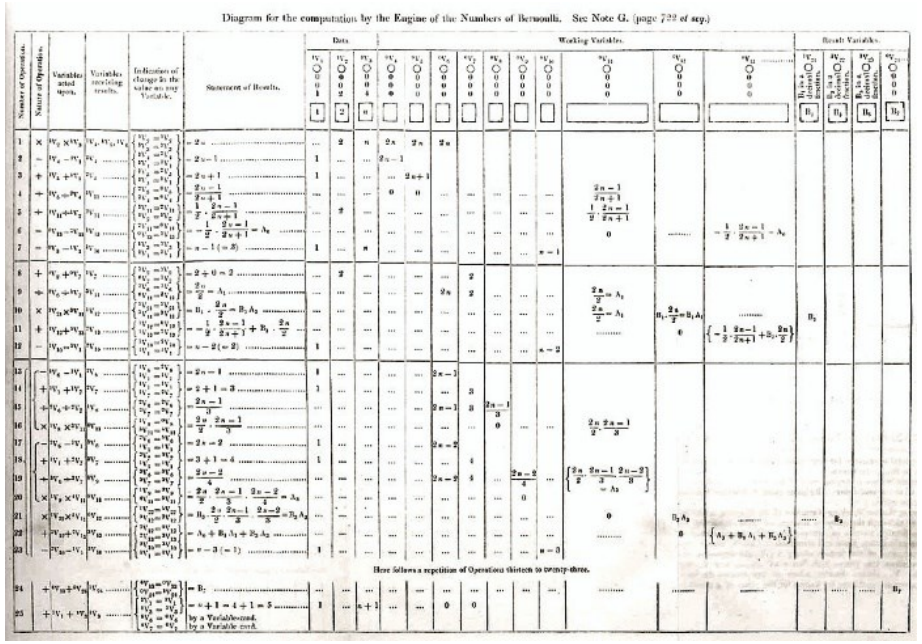
CURIOSITIES

Beginning of High-Level Languages

https://notabletechnicalwomen.org/



- **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."

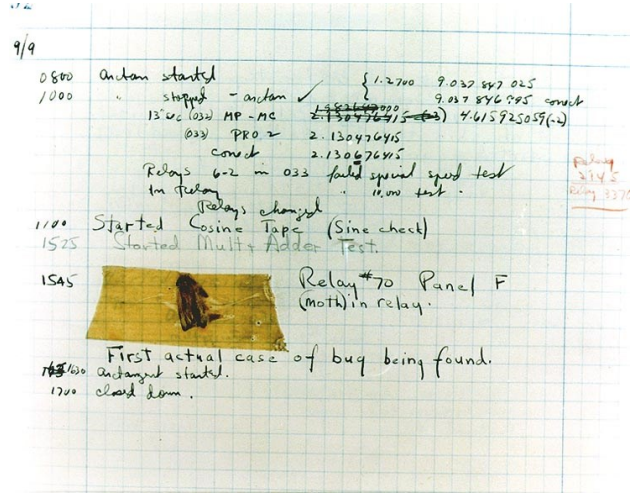


In note G, she describes an algorithm for the Analytical Engine to compute Bernoulli numbers.

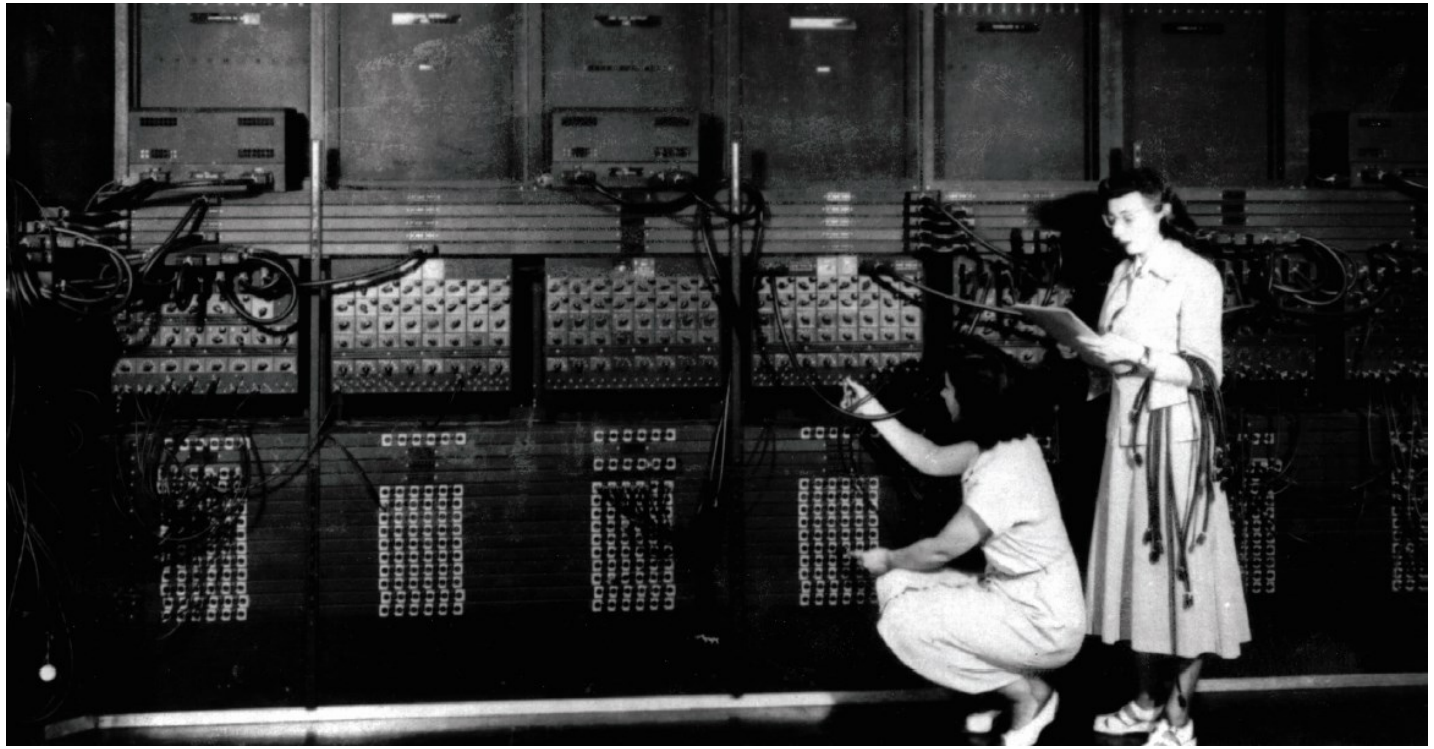
Beginning of the Compiler

https://notabletechnicalwomen.org/

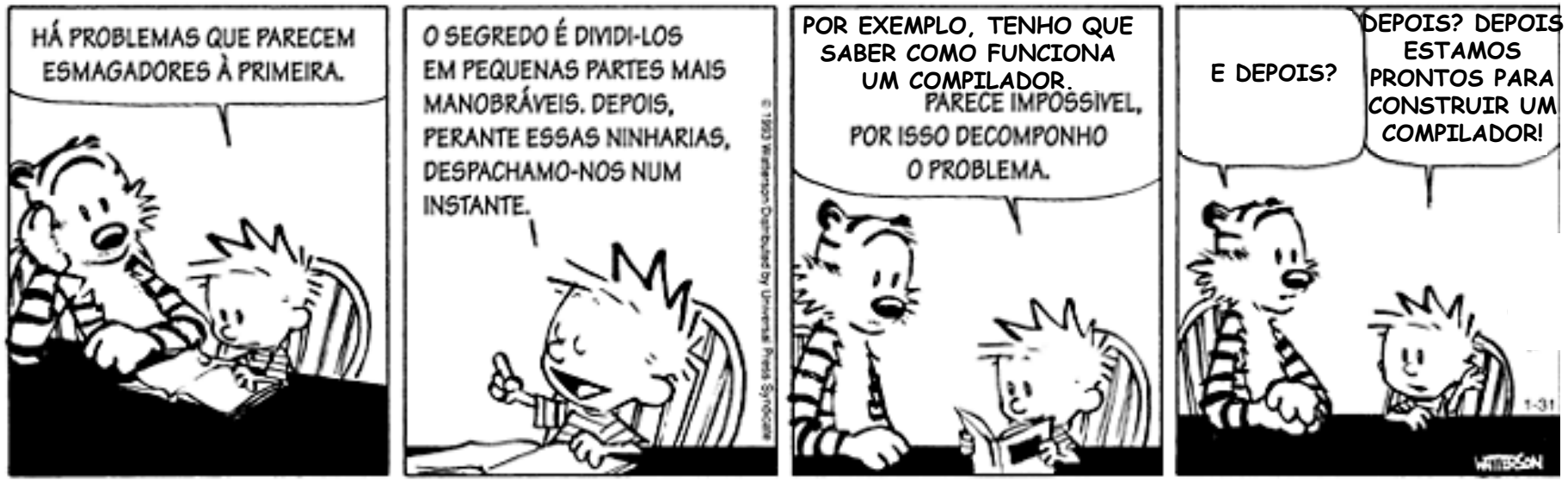
- **Grace Brewster Murray Hopper** (1906 –1992)
  - "One of the first programmers of the Harvard Mark I computer in 1944, invented the first compiler for a computer programming language, and was one of those who popularized the idea of machine-independent programming languages." [source: wikipedia]
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