

Processamento de Linguagem (cod 11567)

Desenho de Linguagens de Programação e de Compiladores (cod. 11482)

Departamento de Informática
Universidade da Beira Interior

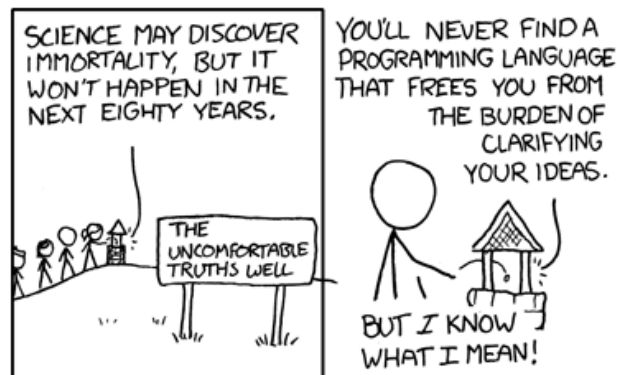


Figura 1: as seen on <http://xkcd.com>

Esta página no formato pdf

1 Novidades

- Notas do exame 1 [aqui \(link\)](#)

- Notas da Avaliação Contínua aqui (link)
- [13/01/2020] As defesas dos trabalhos começam as 14h15 do dia 13 de Janeiro, continuam terça e quarta. A Ficha de inscrição para as defesas está disponível a partir das 9h00 da Segunda-feira 13 de Janeiro no secretariado ou no gabinete do regente. 10 minutos de apresentação, 5 de perguntas.
- [04/01/2020] Notas da Frequência aqui (link)
- [22/10/2019] Aula Teórica de reposição: 4/11/2019 às 18h00 sala 6.26.
- [21/10/2019] : Encontra-se disponível o enunciado do trabalho. Link para a apresentação de João Reis sobre semântica operacional para uma linguagem While (aqui)
- TD1: Se a compilação resultar no erro: `‘relocation R_X86_64_32S against .data’ can not be used when making a PIE object; recompile with -fPIC`, deverá ser usado o comando de compilação `gcc -no-pie ...`
- Como colocar uma dúvida ao regente da Unidade Curricular?
 1. Comparecer nas aulas e colocá-la directamente ao regente
 2. Comparecer no horário de atendimento do regente e colocá-la directamente
 3. enviar um email ao regente (desousaUUU@UUUdi.ubi.pt, (retire os UUU)) com o assunto "DLPC: XXXX" ou "PL: XXXX" em que XXXX é o título da dúvida em questão. Qualquer outro formato no assunto arrisca condenar o email ao esquecimento.
- Inscrição em turmas práticas: via site dos serviços académicos.
- Os alunos com estatuto de *trabalhador estudante* são convidados a dirigir-se ao regente para discutir dos critérios de avaliação.

Conteúdo

1 Novidades

1

2	Docentes	3
3	Objectivos	3
4	Competências por adquirir e Resultados da Aprendizagem	5
5	Programa	7
5.1	Aulas teóricas	7
5.2	Práticas Laboratoriais/ Trabalhos Dirigidos	9
6	Material de Apoio, Pedagógico e Referências Bibliográficas	9
7	CrITÉrios de AvaliaÇão	11
7.1	Componente Ensino/Aprendizagem Prática	11
7.2	Componente Ensino/Aprendizagem Teórica	12
7.3	Admissão e Avaliação por Exame	12
8	Datas Importantes	12
9	Horário	13
10	Atendimento	13

2 Docentes

- Simão Melo de Sousa (regente) - Gabinete 3.17 - Laboratório Release/QuiVVer (6.25) - Bloco VI
- Alexandra Mendes - Gabinete 3.19 - Bloco VI

3 Objectivos

Esta página web visa introduzir a oferta formativa e pedagógica do DIUBI na área disciplinar das linguagens de programação. Assim esta apresenta as principais fases do desenho de linguagens de programação e da construção dum compilador, com ênfase particular nas fases de análise léxica, sintáctica, semântica e de síntese de código para a produção de programas expressivos, eficientes e seguros.

Contexto da Aprendizagem

Na sua componente introdutória apresentamos o conhecimento básico dos fundamentos e das tecnologias das análises léxica, sintáctica, e análises básicas de tipos e de porte. Esta matéria é a componente inicial da UC de *Processamento de Linguagem* do primeiro ciclo em Engenharia Informática da UBI.

Na sequência, procura-se estudar como construir linguagens de programação (e respectivos compiladores) para que esses permitam a expressão ou síntese de programas expressivos, eficiente e bem comportados. Os métodos estudados permitam *calcular* estes programas com as garantias de eficiência e de correcção (por exemplo, relativamente ao código fonte). Assim a *safety*, correcção, eficiência, expressividade, análise comportamental são obtidas por desenho, por cálculo, em tempo de compilação, *automaticamente*. Este é o foco da matéria exposta da UC *Desenho de linguagens de Programação e de compiladores*.

Neste sentido esta componente curricular é complementar da UC de *Computação Fiável - CF* ([link](#)). Ambas estudam a essência das linguagens de programação, dos seus programas, introduzem técnicas relacionadas.

A UC CF visa instruir os seus alunos sobre os conceitos, técnicas, ferramentas e aplicações destas à construção de software fiável e seguro, de programas comprovadamente correctos. Uma introdução a cada família de técnicas é dada mas nem todas são exploradas com todo o detalhe. A abordagem explorada em profundidade nesta UC introduz as famílias de técnicas que permitam obter um perfil comportamental dos programas por análise, raciocínio e demonstração. Por natureza este escrutínio *não é automático* apesar de ser suportado e sistematizado computacionalmente, mas permite uma análise fina, expressiva e complexa.

Para quê estudar os Processos de Compilação?

- Em primeiro porque é interessante e formador!
- Porque as técnicas de processamento de linguagens e de construção de compiladores têm aplicações fortes em muitas áreas da informática, e não só em construção de compiladores.
- Através do estudo da compilação, tentaremos mostrar o quão úteis para a informática em geral são as técnicas e as ferramentas oriundas deste estudo assim como procuraremos perceber as características das

linguagens de programação. De facto um compilador é um programa de tamanho importante que é necessário bem estruturar. Este deve ser também eficiente e é importante que esteja isento de erro. Ser capaz de escrever um compilador é assim uma competência desafio para qualquer programador que se quer exímio.

Agradecimentos

O regente da disciplina gostaria de agradecer

- Jean-Christophe Filliâtre (LRI - Paris Saclay / CNRS, França) por lhe ter facultado o material pedagógico da unidade curricular “Langages de programmation et compilation” de que é autor e regente, por lhe permitido um uso livre deste último.
- Anders Møller pela sua autorização em usar livremente o material pedagógico da unidade curricular Static Program Analysis.

Contexto e parcerias industriais/académicas

Esta UC contou na sua organização e leccionação com vários intervenientes industriais e académicos que citamos aqui (figura 2) como indicador da relevância e abertura desta UC ao meio tecnológico no qual evolui e o seu compromisso firme e reconhecido em potenciar os seus alunos junto desta.

Estes intervenientes influenciaram, participaram na definição da componente prática, propuseram extensões a esta componente na forma de estágios, teses de mestrado, ou contratações. Estas parcerias justificaram a dinâmica escolhida e impressa na exposição teorica e prática da matéria, colaboraram em termos de investigação com a equipa docente em temáticas abordadas nesta UC o que resultou numa exposição que se pretendeu mais esclarecida.

4 Competências por adquirir e Resultados da Aprendizagem

Os estudantes deverão adquirir as seguintes competências:

- Conceber, planear, desenhar e implementar em software processadores de linguagens artificiais e de informação especificada textualmente segundo determinadas regras lexicais e sintáticas;



Figura 2: parceiros e intervenientes (ordem alfabética)

- Conceber e implementar em software as várias etapas relacionadas com compiladores, nomeadamente:
 - expressões regulares e autómatos finitos;
 - analisadores sintácticos;
 - analisadores semânticos.
- Conceber front end e back-ends de compiladores, sistemas de tipo poderosos e modernos, optimizadores de código;
- Conceber, planear, desenhar e implementar linguagens de programação;
- Conceber e implementar em software as várias etapas relacionadas com a construção de compiladores, perceber em que medida podem ser usadas fora do contexto da compilação;
- Concerber desenhar e implementar analisadores estáticos de programas para a optimização e o controlo comportamental de programas (segurança, perfil, depuração, optmização, etc.);
- Perceber os detalhes internos das linguagens de programação e dos seus compiladores como a optimização de código, as implicações técnicas impostas pelas características próprias da linguagem de programação, ou do paradigma na qual se inscreve.

5 Programa

5.1 Aulas teóricas

Revisão OCaml. O seguinte curso poderá ser consultado [Curso OCaml \(link\)](#)

Matéria lecionada. As aulas são organizadas da seguinte forma:

- Aula 1 - *Assembly* x86-64 - exemplo das N rainhas (ficheiro c) (ficheiro assembly)
- Aula 2 - Sintaxe abstracta, semântica formal, interpretadores
- Aula 3 - Tipagem, sistemas e algoritmos

- Aula 4 - Análise léxica
- Aula 5 - Análise sintáctica (descendente) - ficheiro com o código do analisador exposto nas aulas (aqui)
- Aula 6 - Análise sintáctica (ascendente)
- Aula 7 - Compilação de linguagens imperativas, modos de passagem de parâmetros
- Aula 8 - Compilação das linguagens funcionais
- Aula 9 - Compilação das linguagens orientadas a objectos
- Aula 10 - Alocação de memória
- Aula 11 - Produção de código eficiente - parte 1
- Aula 12 - Produção de código eficiente - parte 2
- Aula 13 - Ordens, reticulados e a *framework* monótona
- Aula 14 - Análise de fluxo de dados
- Aula 15 - *Path Sensitivity*, análise interprocedural e análise de fluxo de controlo
- Aula 16 - Análise de apontadores

Nestas aulas, os tópicos seguintes serão abordados (em inglês):

Compilers front-end, Lexical Analysis, (Ascendent and Descendent) Syntactical Analysis, Operational Semantics, Denotational Semantics, Type Checking and (polymorphic) Type Systems, Activation Records, Translation to Intermediate Code, Basic Blocks and Traces, Instruction Selection, Liveness analysis, Register allocation, Garbage collection, Object-oriented languages, Functional Programming Languages, Loop Optimizations, Code Optimizations, Static Single-Assignment Form, Dataflow Analysis, Control Flow Analysis, Pointer Analysis and Other Static Program Analysis. Monotone Framework, Unification Framework, Cubic Framework.

5.2 Práticas Laboratoriais/ Trabalhos Dirigidos

- TD 1 - Assembly X86-64
- TD 2 - Interpretador Mini-Python
- TD 3 - Inferência de tipos e Algoritmo W
- TD 4 - Construção de autómatos deterministas a partir de expressões regulares
- TD 5 - Análise Descendente
- TD 6 - Análise sintática de uma pequena linguagem
- TD 7 - Produção de Código
- TD 8 - GC *stop & Copy*
- TD 9 - Coloração de grafos
- TD 10 - Análise estática de programas na *Framework Monótona*
- TD 11 - A segurança de software como uma análise estática de programas

(new!) enunciado do trabalho aqui ([link](#)).

6 Material de Apoio, Pedagógico e Referências Bibliográficas

Apontamentos apresentados (e disponibilizados) nas aulas.

Um vídeo para acompanhar a mensagem da aula 3 aqui ([link](#)).

Artigos referidos nos acetatos da aula 13: Framework Monótona e Artigo Rice.

Ferramentas

- Assembly x86-64.
 - Página de Andrew Tolmach ([link](#)). Estas notas ([link](#)) em particular.
 - O debugger Nemiver ([link](#)) funciona sobre código x86-64 (execução passo a passo, visualização dos registos, etc.)
 - O site godbolt.org permite comparar facilmente código assembly produzido por diversos compiladores. Existe também uma implementação para o Emacs (<https://github.com/emacsmirror/rmsbolt>).
- Assembly MIPS.
 - Simulador MARS ([link](#)) (aconselhado).
 - Simulador SPIM ([link](#)).
 - um simulador online de MIPS com visualização dos registos VisualMIPS ([link](#));
 - A documentação do SPIM ([link](#)) descreve o conjunto de instruções.
 - um módulo OCaml para escrever e gerar código mips (ml - mli [link](#))
- O gerador de parsers LR(1) Menhir ([link](#)), manual ([link](#)).

UCs Semelhantes e complementares

Apresenta-se aqui apontadores para 3 unidades curriculares que serviram de modelo a UC aqui definida (e cuja exploração se recomenda):

- Langages de programmation et compilation
- Static Program Analysis
- Semantics and applications to verification

Bibliografia Principal

As seguintes obras cobram em profundidade e complementam os tópicos abordados nesta aula.

- Andrew A. Appel. Modern Compiler Implementation in ML.
- A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, Compilers: Principles, Techniques, and Tools, 2nd edition, Addison Wesley, 2006. ISBN 0-321-48681-1.
- Benjamin C. Pierce. Types and Programming Languages.
- Flemming Nielson, Hanne R. Nielson, and Chris L. Hankin. Principles of Program Analysis.
- Hanne. R. Nielson and Flemming Nielson. Semantics with Applications - an appetizer.
- Anders Møller and Michael I. Schwartzbach. Static Program Analysis.
- Randal E. Bryant et David R. O'Hallaron. Computer Systems: A Programmer's Perspective.

7 Critérios de Avaliação

A avaliação avaliará a aprendizagem teóricas e prática dos conceitos introduzidos. Como tal, esta será constituída por **provas escrita** e por um **projecto prático de construção de um pequeno compilador**.

Fraudes A equipa docente gostaria de realçar que qualquer tipo de fraude em qualquer dos itens desta disciplina implica a reprovação automática do aluno faltoso, podendo ainda vir a ser alvo de processo disciplinar. Listamos a seguir as diferentes componentes da avaliação.

7.1 Componente Ensino/Aprendizagem Prática

- A avaliação da Componente Ensino/Aprendizagem Prática mede em termos práticos a aquisição dos conceitos expostos. Como tal é baseada na realização de um **projecto prático de construção de um**

pequeno compilador em grupo de, no máximo, duas pessoas e entregue à equipa docente.

- A *Nota da Componente Prática* (NCP, 20 valores) é a nota do trabalho.

7.2 Componente Ensino/Aprendizagem Teórica

- Esta componente mede em termos teóricos a aquisição dos conceitos expostos. Como tal é baseada na realização de **provas escritas** agendadas no fim de cada capítulo exposto.
- Da média destas provas resulta a *Nota da Componente Teórica* (NCT, 20 valores).

7.3 Admissão e Avaliação por Exame

- Mínimos: são seguidas as disposições aprovadas pelo Conselho Científico da Universidade aplicadas individualmente as duas componentes de avaliação. É admitido a exame quem tiver ambas as notas *NCP* e *NCT* acima dos mínimos (i.e. $NCP \geq 6$ e $NCT \geq 6$).
- A Nota da Prova Escrita do exame substituirá a Nota da Componente Teórica. No final, para obter aprovação a disciplina, esta nota terá de ser maior do que 6. Ou seja:
- A nota final (NF) é calculada pela fórmula:

$$NF = if (NCT \geq 6) then \frac{NCT + NCP}{2} else \textit{Reprovado}$$

8 Datas Importantes

- Entrega do trabalho: última semana de aulas
- Frequência: dia 09/12 de 2019 das 18h00 às 20h00. aula teórica).
- Exame Época 1 : conferir nos SA.
- Exame Época 2 : conferir nos SA.
- Exame Época Especial : conferir nos SA.

9 Horário

Tipo de aula	Horário	Sala
Teórica	Terça-Feira das 16h00 às 18h00	6.26
Prática	Quarta-Feira das 14h00 às 16h00	6.13
Prática	Quarta-Feira das 16h00 às 18h00	6.13
Prática	Quinta-Feira das 09h00 às 13h00	6.13

10 Atendimento

Horário
Terça das 11h00 às 13h00 e das 14h30 às 16h00

ou por mail (medida anti spam, retire os UUU): `desousaUUU@UUUdi.ubi.pt`.

Referências

- [1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1985.
- [2] A. W. Appel. *Modern Compiler Implementation in ML*. Cambridge University Press, 1998.
- [3] Andrew W. Appel, Robert Dockins, Aquinas Hobor, Lennart Beringer, Josiah Dodds, Gordon Stewart, Sandrine Blazy, and Xavier Leroy. *Program Logics for Certified Compilers*. Cambridge University Press, New York, NY, USA, 2014.
- [4] E. Chailloux, P. Manoury, and B. Pagano. Developing applications with objective caml. <http://caml.inria.fr/oreilly-book>, 2003.
- [5] Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, New York, NY, USA, 2012.
- [6] Yaron Minsky, Anil Madhavapeddy, and Jason Hickey. *Real world OCaml*. Sebastopol, Calif. O'Reilly Media, 2013. Index.
- [7] J. Mitchell. *Foundation for Programming Languages*. Foundations of Computing, MIT Press, 1996.

- [8] John C. Mitchell. *Concepts in programming languages*. Cambridge University Press, 2003.
- [9] Steven S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.
- [10] F. Nielson, H. R. Nielson, and C. L. Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999.
- [11] H. R. Nielson and F. Nielson. *Semantics with Applications*. John Wiley & Sons, Chichester, 1993.
http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html
- [12] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, Cambridge, MA, USA, 2002.
- [13] Benjamin C. Pierce. *Advanced Topics in Types and Programming Languages*. The MIT Press, 2004.
- [14] R. Wilhelm and D. Maurer. *Compiler Design*. Addison Wesley, 1995.
- [15] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. Foundations of Computing series. MIT Press, Cambridge, Massachusetts, February 1993.

Enviar comentários e dúvidas para (retire os UUU) : desousaUUU@UUUdi.ubi.pt