Compilers (COMP0012)

Description

Aims:
This is a practical module whose primary goal is develop an understanding of the operation of compilers and the development and specification of computer-based languages. The course pulls together threads from underlying theory, most notably from logic and from data structures and algorithms, and builds on these a practical exercise in which students create a compiler of their own using commonly available compiler development tools.

Learning outcomes:
On successful completion of the module, a student will be able to:
1. build lexical analysers and use them in the construction of parsers;
2. express the grammar of a programming language;
3. build syntax analysers and use them in the construction of parsers;
4. perform the operations of semantic analysis; build a code generator;
5. discuss the merits of different optimisation schemes.

Content:
Anatomy of a compiler:
- The importance of compilers;
- Structure of a compiler;
- Analysis (lexical, syntax and semantic analysis);
- Synthesis (intermediate code generation, optimisation and code generation);
- Compilers vs. interpreters. Lexical analysis (scanning):
  - Tokens;
  - Regular expressions;
- Finite state automata (deterministic and non-deterministic);
- Translating regular expressions into finite state automata;
- Automatic lexer generators (JLex/JFlex). Syntax analysis (parsing):
  - Context-free grammars;
  - Derivations and (concrete/abstract) syntax trees;
  - Handling ambiguous grammars;

Key information

Year: 2019/20
Credit value: 15 (150 study hours)
Delivery: UG L4, Campus-based
Reading List: View on UCL website
Tutor: Dr Earl Barr
Term: Term 2
Timetable: View on UCL website

Assessment

- Written examination (main exam period): 80%
- Coursework: 10%
- Coursework: 10%

Find out more

For more information about the department, programmes, relevant open days and to browse other modules, visit ucl.ac.uk

Disclaimer: All information correct as of June 2019. Please note that aspects of the module may be subject to change. UCL will make best efforts to inform applicants of major changes.
- Top-down parsing (LL(k) grammars, recursive descent parsers);
- Bottom-up parsing (LR(k) grammars, shift-reduce parsers);
- Automatic parser generators (CUP);
- Syntactic error recovery. Syntax-directed translation:
  - Syntax-directed definitions;
  - Abstract syntax tree construction.

Semantic analysis:
- Symbol table management;
- Scoping and type checking;
- Basic implementation techniques (Visitor methodology).

Intermediate code generation:
- Three address code;
- IR instructions;
- Translation methodologies.

Code generation and optimisation:
- Run-time storage organisation;
- A simple code generation algorithm;
- Optimisation of intermediate code;
- Optimisation of target code (Peephole optimisation).

Requisites:
In order to be eligible to select this module, a student must be registered on a programme for which it is a formally-approved option or elective choice.
Description

Aims:
This is a practical module whose primary goal is develop an understanding of the operation of compilers and the development and specification of computer-based languages. The course pulls together threads from underlying theory, most notably from logic and from data structures and algorithms, and builds on these a practical exercise in which students create a compiler of their own using commonly available compiler development tools.

Learning outcomes:
On successful completion of the module, a student will be able to:
1. build lexical analysers and use them in the construction of parsers;
2. express the grammar of a programming language;
3. build syntax analysers and use them in the construction of parsers;
4. perform the operations of semantic analysis; build a code generator;
5. discuss the merits of different optimisation schemes.

Content:
Anatomy of a compiler:
- The importance of compilers;
- Structure of a compiler;
- Analysis (lexical, syntax and semantic analysis);
- Synthesis (intermediate code generation, optimisation and code generation);
- Compilers vs. interpreters. Lexical analysis (scanning):
  - Tokens;
  - Regular expressions;
  - Finite state automata (deterministic and non-deterministic);
  - Translating regular expressions into finite state automata;
  - Automatic lexer generators (JLex/JFlex). Syntax analysis (parsing):
    - Context-free grammars;
    - Derivations and (concrete/abstract) syntax trees;
    - Handling ambiguous grammars;

Key information

Year: 2019/20
Credit value: 15 (150 study hours)
Delivery: UG L5, Campus-based
Reading List: View on UCL website
Tutor: Dr Earl Barr
Term: Term 2
Timetable: View on UCL website

Assessment

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>80%</td>
<td>Coursework: 10%</td>
<td>Coursework: 10%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Find out more

For more information about the department, programmes, relevant open days and to browse other modules, visit ucl.ac.uk

Disclaimer: All information correct as of June 2019. Please note that aspects of the module may be subject to change. UCL will make best efforts to inform applicants of major changes.
Top-down parsing (LL(k) grammars, recursive descent parsers);
Bottom-up parsing (LR(k) grammars, shift-reduce parsers);
Automatic parser generators (CUP);
Syntactic error recovery. Syntax-directed translation:
Syntactic-directed definitions;
Abstract syntax tree construction.

Semantic analysis:
Symbol table management;
Scoping and type checking;
Basic implementation techniques (Visitor methodology).

Intermediate code generation:
Three address code;
IR instructions;
Translation methodologies.

Code generation and optimisation:
Run-time storage organisation;
Simple code generation algorithm;
Optimisation of intermediate code;
Optimisation of target code (Peephole optimisation).

Requisites:
In order to be eligible to select this module, a student must be registered on a programme for which it is a formally-approved option or elective choice.