F29LP Language Processors

**COURSE DETAILS**

**Course Code:** F29LP  
**Full Course Title:** Language Processors  
**SCQF Level:** 9  
**SCAF Credits:** 15  
**Available as Elective:** No

**DELIVERY LEVEL**

Undergraduate: No  
Postgraduate Taught: No  
Postgraduate Research: No

Additional Information:

**COURSE AIMS**

To develop skills in programming language definition

To develop skills in programming language implementation

**LEARNING OUTCOMES – SUBJECT MASTERY**

- Ability to construct & manipulate lexical & syntactic definitions
- Understanding of relationship between grammars & automata
- Ability to construct lexical & syntactic analysers from definitions
- Familiarity with static & dynamic semantics, and associated formalisms
- Ability to generate & traverse ASTs
- Ability to construct AST-based analysers, interpreters & code generators for simple languages
- Understanding of run-time environments
- Ability to use compiler construction tools

**LEARNING OUTCOMES – PERSONAL ABILITIES**

- Appreciation of problem solving as language definition & implementation
- Ability to apply appropriate language oriented formalisms, tools & techniques in solving problems
- Skills in engineering complex software artefacts

**SYLLABUS**

Language definition:

- lexicon: non-terminal & terminal symbols, type 3 grammars, regular expressions, finite state automata, Moore & Mealey machines, left & right recursion
- concrete syntax: type 2 grammars & BNF, factoring grammars, converting left to right recursion, LL(K) & LR(K) grammars, push down automata, parsing
- abstract syntax: abstract syntax from concrete syntax
F29LP Language Processors

- static semantics: types
- dynamic semantics: overviews of axiomatic, denotational & operational semantics; introduction to structural operational semantics (SOS) for declarations, expressions, statements

Language implementation:

- lexical analysis
- syntax analysis: recursive descent for LL(1)
- abstract syntax tree (AST) construction
- AST pretty printing
- overview of interpreter/abstract machine
- AST interpretation
- static analysis of AST: e.g. type checking, identifier resolution
- code generation e.g.: register/memory allocation, structured constructs as test/jump, structured data as structured memory sequences, stack discipline for blocks/subroutines, parameter passing
- run time environment: e.g. heap allocation, garbage collection, debugging support
- compiler construction tools e.g. Lex, Yacc, JavaCC

Based on simple imperative language

<table>
<thead>
<tr>
<th>COURSE RELATIONSHIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course Code</td>
</tr>
<tr>
<td>F28PL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOCATION AND ASSESSMENT METHODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edi</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
</tr>
</tbody>
</table>