

## **MODULE                      PROGRAMMING & COMPILER THEORY**

<b>CODE</b>	BSCH-4-2-09
<b>STAGE</b>	IV
<b>NUMBER OF CREDITS</b>	4 semester credits / 6 quarter units
<b>STATUS</b>	CORE
<b>THEMES</b>	Software Development Computer Systems
<b>ASSESSMENT</b>	Continuous Assessment      30% Examination                      70%

### **Aims**

This module aims to provide the student with an understanding of the underlying mechanisms and theory behind programming languages. As a result of the skills learnt in this module the student will be able to design and write lexers and parsers and be able to design simple programming languages and write compilers for them. Students will also gain an in-depth knowledge of the internal workings of one interpretive compiler/VM via a case study (eg Java or the MS .Net package).

### **Learning Outcomes**

On completion of this module, students will be able to:

- Lexically analyse and parse text files.
- Design simple languages.
- Build compilers for these languages.
- Criticise and categorise programming languages and compilers.
- Explain theoretical capabilities and limitations of present-day compilers.

### **Indicative Content**

<b>Topic</b>	<b>Description</b>
<b>Programming Languages History</b>	A brief overview of programming languages from programming first computers to today's languages.
<b>Categorising translators and compilers.</b>	Translators, Compilers, Interpreters, Interpretive Compilers, Emulators and developing for embedded systems. T-diagrams to describe translators.

<b>Lex</b>	Lexing, mechanisms, motivations. How to use Lex to generate a Lexer. Understanding the Lex utility. Using Lex versus writing customised lexers.
<b>Grammars</b>	Parsing, abstract syntax trees, symbol tables. Context Free Grammars. LL(1) parsing. Limitations of LL(1) parsing. LL(k) parsing. Alternative parsing methods.
<b>Yacc</b>	How to generate a parser using Yacc. Communication between Lex and Yacc. Shift-reduce paradigm, symbol and value stacks. Resolving grammatical conflicts (shift-reduce and reduce-reduce).
<b>Case Study</b>	An in depth look at an interpretive compiler/VM architecture such as Java or .Net with Rotor (Shared Source Common Language Infrastructure)

### Teaching and Learning Methods

Students will be taught using a combination of lectures and tutorials. Tutorial sessions will be based on worksheets. These will contain small practical exercises relating to Lex and Yacc and will give the students a good grounding for their programming assignment.

### Assessment Methods

Students will be graded on the basis of a programming assignment and an end of semester examination. The programming assignment will typically cover such tasks as building small compilers for 'hobby-type' languages.

### Primary Reading List

<b>Title</b>	<b>Author</b>	<b>Publisher</b>
Compilers: Principles, Techniques, and Tools by (the Red Dragon Book)	Aho, Sethi, and Ullman	Addison-Wesley 1986
Lex & Yacc, 2nd edition	Levine, Mason, Brown	O'Reilly & Associates, Inc 1992

### Recommended Reading List

<b>Title</b>	<b>Author</b>	<b>Publisher</b>
Programming Language Pragmatics	Michael L. Scott	Morgan Kaufmann Publishers 2000
Modern Compiler Implementation in Java (2 <sup>nd</sup> Edition)	Andrew W. Appel	Cambridge University Press 2002

Inside the Java Virtual Machine	Bill Venners	McGraw-Hill 1998
Distributed Programming Runtime Systems: Inside Rotor (Draft Copy)	Gary Nutt	Addison Wesley 2003