University of Macau
Faculty of Science and Technology
Department of Computer and Information Science
SFTW341 Compiler Construction
Syllabus
1st Semester 2012/2013
Part A – Course Outline

Compulsory course in Computer Science

Course description:
(4-1) 4.5 credits. Modern compiler design, use of automatic tools, compilation techniques and program intermediate representations; scanner, recursive descent parser, bottom-up parser, code generation and optimization; semantic analysis and attribute grammars, transformational attribute grammars.

Course type:
Theoretical with substantial laboratory/practice content

Prerequisites:
- SFTW223

Textbook(s) and other required material:
- David A Watt and Deryck F Brown. (2000) *Programming Language Processors in JAVA — Compilers and Interpreters***.** Prentice Hall, US.

References:
- Reis, A. J. D. (2011). *Compiler Construction Using Java, JavaCC, and Yacc*. Wiley-IEEE Computer Society Pr.
- Alfred V Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D Ullman. (2007). *Compiler: Principles, Techniques and Tools*. 2nd ed., Prentice Hall.
- Charles N. Fischer, Ron K. Cytron, and Richard J. LeBlanc. (2010). *Crafting A Compiler*. Pearson Higher Education.

Major prerequisites by topic:
- Programming languages and algorithms
- Formal languages and finite state automata
- Regular expressions and grammars
- Logical operations

Course objectives:
- Learn the detailed operation of the major phases of a compiler [a, c]
- Introduce the theory behind the various phases, including regular expressions, context free grammars, and finite state automata [a, e]
- Apply the theoretical foundations that form the basis of compilation [a, e]
- Design and implement parts of a compiler for a small imperative-style programming language [a, c, e, k]
- Practice software engineering design principles on a medium size project [a, c, e, k, l]

Topics covered:
- **Basic Concepts (7 hours)**: Review the concepts of high-level programming languages, and their syntax, contextual constrains and semantics, with examples from well-known programming systems. Introduce basic terminology of language processors: translators, compilers, interpreters, source and target languages, and real and abstract machines. Study the way of using language processors with Tombstone diagram.
- **Theoretical Foundations (6 hours)**: Review the fundamentals of formal language concepts, including finite-state automata, regular expressions, construction of equivalent deterministic finite-state automata from regular expressions, context-free grammars, grammar notation, derivations, and parse trees.

- **Syntactic Analysis (8 hours)**: Study the details of syntactic analysis, including scanning, parsing, and abstract syntax tree construction. Introduce recursive-descent parsing techniques, and show how a parser and scanner can be systematically constructed from source language's syntactic specification.
- **Contextual Analysis (8 hours)**: Study the details of contextual analysis module, in case of that the source language exhibits static bindings and is statically typed. Introduce the techniques to validate identifier which is related to language's scope rules, and type checking which is related to language's type rules.
- **Run-Time Organization (8 hours)**: Discuss the relationship between the source language and the target machine. Show how target machine instructions and storage must be marshaled to support the higher-level concepts of source language. The topics include data representation, expression evaluation, storage allocation, routines and their arguments, garbage collection, and run-time organization of simple object-oriented languages.
- **Code Generation (12 hours)**: Study the details of code generation. Show how to organize the translation from source language to object code. It relates the selection of object code to the semantics of source language in a stack-based machine. Basic techniques of code optimization are introduced at different phases: profiler optimization, intermediate code optimization and target code optimization.
- **Interpreters & Compiler Tools (6 hours)**: Look inside interpreters. It gives examples of interpreters for both low-level and high-level languages, as well as introduces the compiler construction tools: Lex & Yacc.

Class/laboratory schedule:

Timetabled work in hours per week			**No of teaching weeks**	**Total hours**	**Total credits**	**No/Duration of exam papers**
Lecture	**Tutorial**	**Practice**				
4	1	Nil	14	70	4.5	1 / 3 hours

Student study effort required:

Class contact:	
Lecture	56 hours
Tutorial	14 hours
Other study effort	
Self-study	42 hours
Homework assignment	9 hours
Project / Case study	15 hours
Total student study effort	136 hours

Student assessment:
Final assessment will be determined on the basis of:

Homework	10%	Project	20%
Mid-term	30%	Final exam	40%

Course assessment:
The assessment of course objectives will be determined on the basis of:
- Homework, project and exams
- Course evaluation

Course outline:

Weeks	**Topic**	**Course work**
1-2	**Introduction** Specification of programming language, language processors, Tombstone diagrams, bootstrapping, architecture of compiler, different analytical phases	
3	**Theoretical Foundations** Finite-state automata, regular expression, context-free grammar	Course Project
4-5	**Syntactic Analysis** Grammar transformation, parsing strategy, development of recursive-descent parser, intermediate representation (abstract syntax trees), scanner and error handling	Assignment#1

Weeks	Topic	Course work
6-7	**Contextual Analysis** Organization of identification, type & scope checking, analysis algorithm	Assignment#2
8-9	**Run-Time Organization** Data representation, expression evaluation, storage allocation, routines and heap storage allocation	Midterm exam
10-11	**Code Generation** Code function, code template, generation algorithm, manipulation of constants & variables, procedures & functions	Assignment#3
12	**Code Optimization** Basic block, flow graph, local optimization, global optimization, peephole optimization, and loop optimizations	
13	**Interpretation** Interactive interpretation, recursive interpretation **Compiler Construction Tools** Lex & Yacc	
14	**Project Demonstration**	

Contribution of course to meet the professional component:
This course prepares students with fundamental knowledge and experiences to constructing a language processor.

Relationship to CS program objectives and outcomes:
This course primarily contributes to the Computer Science program outcomes that develop student abilities to:
(a) an ability to apply knowledge of mathematics, science, and engineering.
(c) an ability to design a system, component, or process to meet desired needs within realistic constraints such as economic, environmental, social, political, ethical, health and safety, manufacturability, and sustainability.
(e) an ability to identify, formulate, and solve engineering problems.
(k) an ability to use the techniques, skills, and modern engineering tools necessary for engineering practice.

The course secondarily contributes to the Computer Science program outcomes that develop student abilities to:
(l) an ability to use the computer/IT tools relevant to the discipline along with an understanding of their processes and limitations.

Relationship to CS program criteria:

Criterion	DS	PF	AL	AR	OS	NC	PL	HC	GV	IS	IM	SP	SE	CN
Scale: 1 (highest) to 4 (lowest)		4	2				1						2	

Discrete Structures (DS), Programming Fundamentals (PF), Algorithms and Complexity (AL), Architecture and Organization (AR), Operating Systems (OS), Net-Centric Computing (NC), Programming Languages (PL), Human-Computer Interaction (HC), Graphics and Visual Computing (GV), Intelligent Systems (IS), Information Management (IM), Social and Professional Issues (SP), Software Engineering (SE), Computational Science (CN).

Course content distribution:

Percentage content for			
Mathematics	Science and engineering subjects	Complementary electives	**Total**
0%	100%	0%	100%

Coordinator:
Prof. Xiao Shan Li

Persons who prepared this description:
Dr. Fai Wong

Part B – General Course Information and Policies

1st Semester 2012/2013

Instructor:	Dr. Fai Wong	Office:	R108
Office hour:	Mon ~ Fri 11:00 am – 13:00 pm, or by appointment	Phone:	8397 8051
Email:	derekfw@umac.mo		

Time/Venue: Mon 8:30 – 10:30, L108 (lecture)
Thu 8:30 – 10:30, L108 (lecture)
Sat 13:30 – 15:30, J418 (tutorial)

Grading distribution:

Percentage Grade	Final Grade	Percentage Grade	Final Grade
100 - 93	A	92 - 88	A–
87 - 83	B+	82 - 78	B
77 - 73	B–	72 - 68	C+
67 - 63	C	62 - 58	C–
57 - 53	D+	52 - 50	D
below 50	F		

Comment:
The objectives of the lectures are to explain and to supplement the text material. Students are responsible for the assigned material whether or not it is covered in the lecture. Students who wish to succeed in this course should read the textbook prior to the lecture and should work all homework and project assignments. You are encouraged to look at other sources (other texts, etc.) to complement the lectures and text.

Homework policy:
The completion and correction of homework is a powerful learning experience; therefore:
- There will be approximately 3 homework assignments.
- Homework is due one week after assignment unless otherwise noted, no late homework is accepted.
- Homework must be neatly typed and printed on a word-processor.
- The course grade will be based on the average of the homework grades.

Course project:
The project is probably the most exciting part of this course and provides students with meaningful experience to extend and enhance an existing compiler and interpreter:
- You will work with group of two students for the course project.
- The requirements will be announced and discussed in class.
- The project will be presented at the end of semester.

Exams:
One 2-hour mid-term exam will be held during the semester. Both the mid-term and final exams are closed book examinations. There will be occasional in-class assignment.

Note:
- Check UMMoodle (https://ummoodle.umac.mo/) for announcement, homework and lectures. Report any mistake on your grades within one week after posting.
- No make-up exam is given except for CLEAR medical proof.
- Cheating is absolutely prohibited by the university.

Appendix:

Rubric for Program Outcomes

Rubric for (a)	**5 (Excellent)**	**3 (Average)**	**1 (Poor)**
Understand the theoretic background	Students understand theoretic background and the limitations of the respective applications.	Students have some confusion on some background or do not understand theoretic background completely.	Students do not understand the background or do not study at all.
Rubric for (c)	**5 (Excellent)**	**3 (Average)**	**1 (Poor)**
Design capability and design constraints	Student understands very clearly what needs to be designed and the realistic design constraints such as economic, environmental, social, political, ethical, health and safety, manufacturability, and sustainability.	Student understands what needs to be designed and the design constraints, but may not fully understand the limitations of the design constraints.	Student does not understand what needs to be designed and the design constraints.
Rubric for (e)	**5 (Excellent)**	**3 (Average)**	**1 (Poor)**
Identify applications in engineering systems	Students understand problem and can identify fundamental formulation.	Students understand problem but cannot apply formulation, or cannot understand problem.	Students cannot identify correct terms for engineering applications.
Rubric for (k)	**5 (Excellent)**	**3 (Average)**	**1 (Poor)**
Use modern principles, skills, and tools in engineering practice	Student applies the principles, skills and tools to correctly model and analyze engineering problems, and understands the limitations.	Student applies the principles, skills and tools to analyze and implement engineering problems.	Student does not apply principles and tools correctly and/or does not correctly interpret the results.
Rubric for (l)	**5 (Excellent)**	**3 (Average)**	**1 (Poor)**
Use modern computer/IT tools relevant to the discipline	Student uses computer/IT tools relevant to the engineering discipline, and understands their limitations.	Student uses computer /IT tools relevant to the engineering discipline.	Student does not use computer/IT tools relevantly, and does not understand their limitations.