Subject Introduction

COMP3131/9102: Programming Languages and Compilers

Term 1, 2021

FAQs

1. Course Aims

This subject focusses on basic compiler techniques needed to implement programming languages in a virtual machine (with emphasis on the compiler front end). By writing a compiler in Java to translate a variant of C into Java bytecode, you are also expected to gain a good understanding of important fundamental principles in object-oriented programming languages and develop further your programming and software engineering skills.

This project, which consists of a sequence of five programming assignments, can be challenging but fun. It is probably the first time that you write a compiler and run a program that you compiled! In exchange for your hard work, you will learn a lot about programming languages, compilers, software development and how to use programming language expertise to make a large project work better.

Graduate attributes (UNSW graduate attributes are the qualities, skills and understandings a university agrees its students should develop during their studies. Some of these graduate attributes will be developed and assessed in this course in terms of the knowledge and skill outcomes listed below.

Knowledge Outcomes:

- finite state automata and how they relate to regular expressions
- context free grammars and how they relate to context-free parsing
- formal language specification strategies
- top-down and bottom-up parsing styles
- attribute grammars
- type checking
- Java virtual machine
- code generation techniques
- visitor design pattern

Skill Outcomes:

- ability to write scanners, parsers, semantic analysers and code generators
- ability to use compiler construction tools: lexers and parsers
- understand how to specify the syntax and semantics of a language
- understand code generation
- understand and use the data structures and algorithms employed within the compilation process
- ability to write reasonably large OO programs in Java using packages, inheritance, dynamic dispatching and visitor design pattern
- understand virtual machines, in particular Java virtual machine
- Practical ways to take advantage of understanding compilers in software development, design and management
- The deeper insights into the nature of programming languages, software and computation that understanding compilers provide

Here are some ongoing compiler projects (and job opportunities):

- Open Projects - LLVM Compiler

- The Solidity Compiler (for compiling smart contracts written in Solidity)
- Deep Learning Compiler and Optimizer -- Microsoft Research
- Swift for TensorFlow Project Home Page
- MLIR: A new intermediate representation and compiler framework
- Some Awesome Compilers
- compilerjobs.com
- CUDA LLVM Compiler (for GPUs)
- Google's Closure Compiler for Javascripts
- Apple's other open secret: the LLVM Compiler
- Jack (Java Android Compiler Kit)
- Open deep learning compiler stack for cpu, gpu and specialized accelerators

2. Units of Credit

6

3. Assumed Knowledge

Students must have successfully completed COMP2511: Object-Oriented Design and Programming or COMP2911: Engineering Design in Computing before attempting this subject. Therefore, this subject makes some basic assumptions about your knowledge of Java, data structures and algorithms. You are expected to program efficiently.

4. COMP3131/COMP9102

This is a combined undergraduate (COMP3131) and postgraduate (COMP9102) course. The course web pages can be accessed via:

http://www.cse.unsw.edu.au/~cs3131/

or

http://www.cse.unsw.edu.au/~cs9102/

You should become familiar with the structure and content of the subject web pages early on, and consult them frequently. In particular, all important announcements will be posted to the Subject Notice Board on the opening page.

It is your responsibility to check our web pages regularly.

5. Staff

Name	Room	Phone	Contact	Role
Jingling Xue	K17-501K	9385 4889	j.xue@unsw.edu.au	Lecturer-in-charge + Admin
Samad Saadatmand Feyzrasa			samad.saadatmand@gmail.com	Tutor

6. Consultation

There is no fixed consulation time for this course.

You will usually find the answers to most of your questions in the course forum and the FAQs for our programming assignments. If you would really like to discuss something with me individually, past experience shows that a lot of issues can be resolved quickly by email. Please email your

questions to me at my UNSW email address. If you would like to see me in person, please arrangement an appointment with me by email.

7. The Handbook Entry

```
Staff Contact: Professor J. Xue
UC 6 S2 L3 T1
Prerequisites: COMP2011/COMP2711/COMP2911
Equivalent: COMP9102.

Covers the fundamental principles in programming languages and
implementation techniques for compilers (emphasis on compiler
front ends). Course contents selected from: program syntax and semantics,
formal translation of programming languages, finite-state recognisers
and regular expressions, context-free parsing techniques such as
LL(k) and LR(k), attribute grammars, syntax-directed translation,
type checking, code optimisation and code generation.

Lab: implementation of a compiler in a modern programming language
for a small programming language.
```

8. Syllabus

The following is a summary of the topics that will be covered in this subject:

1. Lexical analysis
    - crafting a scanner by hand
    - regular expressions, NFA and DFA
    - scanner generator (e.g.,, lex and JLex)
2. Context-free grammars
    - parsing
    - derivations
    - parse trees
    - abstract syntax trees (ASTs)
3. Syntactic analysis
    - recursive-descent parsing and LL(k) parsing
    - bottom-up parsing and LR(k) parsing
    - parser generators (e.g., yacc, JavaCC and JavaCUP)
4. Semantic analysis
    - attribute grammars
    - symbol table
    - identification (i.e., binding)
    - type checking
5. Code generation
    - syntax-directed translation
    - Jasmin assembly language
    - Java Virtual Machine (JVM)
    - Java byte code generation

9. Reading Material

The following book is the recommended text for this subject:

- Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools*, 2/E, Addison Wesley, 2007. ISBN-10: 0321486811. ISBN-13: 9780321486813.
  This is the Bible on compiler construction. The 1st edition is commonly known as the Red Dragon Book from the picture on its cover. The 2nd edition is the Purple Dragon Book. As far fas this course is concerned, either edition suffices. For the professional, these books are definitely the ultimate references. For the uninitiated,

some chapters can be tough to read. However, not all the topics in the text will be given the same amount of attention, with preference tending towards the topics needed to support the assignment tasks of implementing our VC compiler.

The lecture transparencies and recommended on-line resources over the semester should provide sufficient supplement materials for you to study this subject well.

Other useful reference books include the following:

- Dick Grune, Henri E. Bal, Ceriel J.H. Jacobs and Koen G. Langendoen, *Modern Compiler Design*, John Wiley & Sons, Ltd., 2000. ISBN 0471-976970
- Reinhard Wilhelm and Dieter Maurer, *Compiler Design*, Addison-Wesley, 1995.
- Andew W. Appel, *Modern Compiler Implementation in Java*, Cambridge University Press, 1998.
- Afred V. Aho and Jeffrey D. Ullman, *Theory of Parsing, Translation and Compiling*, Addison-Wesley, 1986.
- C. N. Fisher and R. J. LeBlanc, Jr., *Crafting a Compiler with C*, Benjamin/Cummings, 1991.
- Thomas. W. Parsons, *Introduction to Compiler Construction*, W. H. Freeman and Company, 1992.
- N. Wirth, *Compiler Construction*, Addison-Wesley, 1996.

10. Programming Assignments

Every student will do a compiler project that will produce a compiler that translates a variant of C, called VC, into Java bytecode.

The project must be coded in Java.

The project is broken up into five individual assignments:

1. scanner: read characters and produces tokens
2. recogniser: read tokens and parse the program only for syntactic correctness
3. parser: extend the parser to build an abstract syntax tree (AST)
4. static semantics: check semantics at compile-time
5. code generator -- generate Java bytecode

The completed compiler will have approximately 5,500 lines of code. We will supply many classes to simplify the coding, so you will have to write about between 2,500 and 3,000 lines of code.

By the end of the semester you will have implemented a compiler that will produce code that can be executed.

You will implement each assignment component by building on the previous ones that you have completed. Therefore, you are strongly advised to complete all assignments and complete them on time.

Assignments are to be completed individually and without assistance from others except the staff in this subject. Of course, all legal collaborations and discussions among students are encouraged. Our policies regarding plagiarism are described in Section 15.

There is a bonus project on writing a compiler for an extended language of VC. Details will be available later.

11. Final Exam

There will be a two-hour closed-book written exam for this subject in the June exam period. The exam will consist of mostly questions on the fundamental principles of the subject. For example, the topics that typically appear in the exam include regular expressions, NFA, DFA, context-free grammars, attribute grammars, and LL parsing.

You will not be expected to remember all the details of the JVM instructions. A subset of JVM instructions that will be used in the exam will be provided.

12. Teaching Strategies

This is a course taught now in the first term of a year. There are two two-hour lectures per week, one one-hour tutorial per week (starting from Week 2), five programming assignments and one two-hour final exam.

The course has been designed so that lectures, tutorials and assignments are well integrated to maximise your learning effectiveness. During the first lecture, an typical structure for compilers (or precisely compiler frontends) will be presented, identifying the four major compiler components (scanner, parser, semantic analyser, and code generator). Throughout this course, the subject matter is organised in relation to these components and their interactions. The lectures tend to focus on techniques that fit within a particular component, while the programming project emphasises the implementations of these individual components and the integration of techniques within and across components.

Lectures will include explanations of algorithms and techniques and demonstrations of their dynamic behavior by a small number of examples. Question answering sessions (in lectures) will be intended to provide assistance with the assignments by addressing difficulties that arise as in the course of student implementation work. In tutorials, more examples will be worked out to ensure your understanding of the topics introduced in lectures. Note that UNSW policy stipulates that students attend at least 80% of lectures. While tutorials are not compulsory, it is important that you attend them since compiler algorithms and techniques are mostly new to you and take a few good examples to digest and understand.

This course is project-driven. Each assignment is built on the previous ones so that all five completed assignments will form a Java-to-VC compiler! Therefore, the knowledge that is needed for implementing an assignment will be introduced sufficiently earlier to give you enough time to work on that assignment. The due date for an assignment is set intentionally to be at a weekend. I will strike to finish marking an assignment 1 -- 2 days after the due date so that I can provide students with general feedback for the assignment in the next lecture. This will enable you to fix the bugs in your implementation while it is still fresh in your mind and proceed to work on the next assignment immediately.

Few of you will write conventional compilers but most will need to develop compiler-like processors to solve many application problems in the world. Therefore, the course will build on an understanding of fundamental principles, some of which may not be directly exercised in our five programming assignments.

13. Learning Strategies

The knowledge outcomes listed in Section 1 will be achieved through a combination of lectures, tutorials and programming assignments. This is because the corresponding knowledge units will be introduced (with a few illustrating examples) in lectures, illustrated further by more examples in tutorials and re-enforced in programming assignments. I will use lectures to not only introduce new material but also provide general feedback for assignments and advise for the current assignment. So it is important that you attend all lectures if you can. Note that UNSW policy stipulates that students must attend at least 80% of lectures. While tutorials are not compulsory, it is important that you make an effort to attend them. Weekly tutorials offer help and guidance on both theoretical and practical problems, and allow for more detailed discussion of specific topics as required.

The skill outcomes listed in Section 1 will mostly be achieved during the process of building our Java-to-VC compiler, which is divided into five programming assignments, i.e., five **milestones**. To learn effectively, it is essential that you complete all assignments and complete them on time since each assignment depends on the previous ones. These assignments limit procrastination, and more importantly, allow for reflection about each compiler module and the associated knowledge

units you learned earlier. With the help of my timely feedback, you can spot problems you may have with concepts and implementation.

A reasonable number of test cases are provided for each assignment. However, you will definitely need to create more test cases to ensure your implementation works properly. You will find that you learn more by designing good test cases by yourself.

14. Assessment

14.1. Rationale

Programming assignments have been designed to support you in achieving most of the learning (knowledge and skill) outcomes listed in Section 1. However, the purpose of the final exam is to test your understanding of topics that are not tested or difficult to fully assess in programming assignments and also to allow you to show that the abilities demonstrated in the assignments are their own.

14.2. Criteria

The subject assessment includes components from the compiler project and a final closed-book exam.

The compiler project is marked out of 100 in the following proportions:

Component	Value	Due (Tentative)
Assignment 1	12/100	Week 2
Assignment 2	12/100	Week 4
Assignment 3	18/100	Week 6
Assignment 4	30/100	Week 8
Assignment 5	28/100	Week 10

The final exam is also marked out of 100.

Your final mark in this subject will be based on a combination of the above components.

```
prog = mark for all programming assignments (out of 100)
exam  = mark for the final exam (out of 100)
bonus = mark for the bonus programming assignment (out of 5)

final = min((2 x prog x exam) / (prog + exam) + bonus, 100)
```

14.3. How are Assignments Marked?

All assignments are evaluated on correctness and the bulk of the marking is done by examining the output. We run your program through a large number of test cases and compare (diff) your output to the output of our solution. Your best bet is to thoroughly test your program on the provided test cases and make up others of your own until you feel confident in your solution.

We will not spend a lot of time examining your source and/or giving design feedback. It is not that we don not consider style or design as being important, but by now all of you have taken a course like COMP2511/2911 and we hope you have internalised and practised good coding practices as a

matter of habit. Thoughtful design and coding will improve your chances of having a correctly working program, too.

15. Course Policies

15.1 Email

I will try to answer your questions as soon as possible. Please direct all your questions to the course forum so that the answers provided by me or others can also benefit some other students in the class.

15.2. Lateness

Each assignment is built on the preceding assignments and is typically marked within 48 hours of its submission. All assignment modules will eventually form a compiler that translates a variant of C into Java assembly code. Therefore, if you don't complete an assignment on time, you will usually have difficulty on completing the rest of the assignments on time, particulary since we have now moved to trimesters (with 10 weeks per term)!

Past experience shows that almost all students were able to complete all assignments on time and there were very few late extensions granted each year.

Late programming assignments will be penalised unless you have legitimate reasons to convince me otherwise. For details, see the individual assignment pages.

However, serious personal problems will be dealt with individually.

15.3. Plagiarism

As you should be aware, UNSW has a commitment to detecting plagiarism in assignments. In this particular course, we run a special program that detects similarity between assignment submissions of different students, and then manually inspect those with high similarity to guarantee that the suspected plagiarism is apparent.

If you receive a written letter relating to suspected plagiarism, please contact the LIC with the specified deadline. ***While those students can collect their assignments, their marks will only be finalised after we have reviewed the explanation regarding their suspected plagiarism.***

Here is a statement of UNSW on plagiarism:

Plagiarism is the presentation of the thoughts or work of another as one's own.*

Examples include:

- direct duplication of the thoughts or work of another, including by copying material, ideas or concepts from a book, article, report or other written document (whether published or unpublished), composition, artwork, design, drawing, circuitry, computer program or software, web site, Internet, other electronic resource, or another person's assignment without appropriate acknowledgement;
- paraphrasing another person's work with very minor changes keeping the meaning, form and/or progression of ideas of the original;
- piecing together sections of the work of others into a new whole;
- presenting an assessment item as independent work when it has been produced in whole or part in collusion with other people, for example, another student or a tutor; and,
- claiming credit for a proportion a work contributed to a group assessment item that is greater than that actually contributed.†

Submitting an assessment item that has already been submitted for academic credit elsewhere may also be considered plagiarism. Knowingly permitting your work to be copied by another student may also be considered to be plagiarism. An assessment item produced in oral, not written form, or involving live presentation, may similarly contain plagiarised material.

The inclusion of the thoughts or work of another with attribution appropriate to the academic discipline does *not* amount to plagiarism.

Students are reminded of their Rights and Responsibilities in respect of plagiarism, as set out in the University Undergraduate and Postgraduate Handbooks, and are encouraged to seek advice from academic staff whenever necessary to ensure they avoid plagiarism in all its forms.

The Learning Centre website is the central University online resource for staff and student information on plagiarism and academic honesty. It can be located at:

www.lc.unsw.edu.au/plagiarism

The Learning Centre also provides substantial educational written materials, workshops, and tutorials to aid students, for example, in:

- correct referencing practices;
- paraphrasing, summarising, essay writing, and time management;
- appropriate use of, and attribution for, a range of materials including text, images, formulae and concepts.

Individual assistance is available on request from The Learning Centre.
Students are also reminded that careful time management is an important part of study and one of the identified causes of plagiarism is poor time management. Students should allow sufficient time for research, drafting, and the proper referencing of sources in preparing all assessment items.

* Based on that proposed to the University of Newcastle by the St James Ethics Centre. Used with kind permission from the University of Newcastle.
† Adapted with kind permission from the University of Melbourne.

16. Course Evaluation and Development

This course is evaluated using the myExperience system at the end of each semester. In general, COMP3131/9102 rates reasonably well. According to the student feedback in the past, our programming assignments appear to be easy for some, a bit too hard for some others and about just right for the rest. In my view, our programming assignments are comparable to those used in similar courses offered worldwide. Those who want extra work are encouraged to do the bonus project, which will be released later in the course.

In general, the spec for an assignment will be released immediately after the due date of the previous assignment. However, when we move towards the end of the term, the specs for the last few assignments may be released earlier to give you more time to work on them.

As a combined course for both COMP3131 and COMP9102, our lectures will be held in the evening during 6pm -- 8pm (Monday and Thursday). So our lectures cannot be moved earlier.

If you have any concerns, you may raise them with your lecturer, or approach the School Ethics Officer , Grievance Officer , or one of the student representatives .

I hope you enjoy this subject!

Jingling Xue
Last updated 03/19/2021 11:21:24