Kurdistan Regional Government-Iraq
Ministry of Higher Education and Scientific Research
Universy of Salahaddin – Hawler
College of Engineering
Department of Software Engineering

Course Book

Compilers

3rd Year Material

Academic year (2011-2012)

Prepared by: Lecturer Amanj Mustafa Mahmud, M.Sc.

Email: amanjpro@gmail.com
webpage: http://www.amanj.me

1 Course Description

A compiler translates a computer program from a high level language, such as C, Python or Java, to machine code, the internal representation in the computer. Compilation takes several steps. The first step is lexical analysis, to separate the program into “words”. Syntactic analysis finds the structures. Code generation is often done in two steps, via an intermediate code to machine code. Often the code is improved through code optimization.

The methods and tools from compiler design are useful for other forms of translation, such as from XML to a data structure. The course is taught in English (and the exams should be written in English too). In order to pass the course, students should pass the assignments (all of them) as well as the written exams.

1.1 For whom this course is intended?

This course is intended for the third stagers at Software Engineering undergraduate program.

1.2 Prerequisites of the course (topics)

The students should have finished the courses of the first and second stages to be able to take this course.

1.3 Unusual circumstances of the course

None

2 Objectives

To pass, the student must understand, how simple imperative programming languages equivalent to C can be compiled to machine code for RISC-like machines.
The students must specifically be able to :

2.1 structure a compiler as a sequence of distinct translation steps.
2.2 use regular languages to describe the lexical elements of a programming language.
2.3 describe lexical analysis using a finite automaton.
2.4 use context free languages to describe the syntactic structure of a programming language.
2.5 use the parsing methods top-down (recursive descent) and bottom-up (LR).
2.6 use abstract syntax trees to represent the results of the syntactic analysis.
2.7 break down statements and expressions to simpler designs, and translate syntax trees to intermediate code.
2.8 describe how recursive procedure calls can be implemented by means of stacks, activation posts and machine registers.
2.9 translate the simplified intermediate code of a program to machine-specific instructions.

3 Content and Organization

3.1 Lexical analysis (scanning)
3.2 Syntactical analysis (parsing).
3.3 Program representation in Abstract Syntax Trees (AST).

3.4 Symbol tables and scope rules for C-like languages.
3.5 Type-checking for C-like languages.
3.6 Different forms of intermediate code (IR).
3.7 Generation of intermediate code.
3.8 Call stacks and activation posts
3.9 Code generation for RISC-like machines
3.10 Basic blocks, control-flow graphs, liveness-analysis, register allocation.

4 Teaching Methodology

4.1 22 Lectures, 3 hours/week.
4.2 Four mandatory assignments, students can collect up to 6 bonus points through the assignments.
4.3 Contact hours: 2 hours per week.

5 Evaluation

5.1 There will be four mandatory assignments, each of them weighs 1.5 bonus points, with the following policy:
- Full mark → 1.5 bonus points.
- 9 → 1 bonus points.
- 8 → 0.5 bonus point.
- 7, 6, 5 → pass (but 0 bonus points).
- 4, 3, 2, 1, 0 → resubmit the assignment and 0 bonus points.

5.2 Written exams, three in total: first term exam, second term exam and final exam.
5.3 The final grade is calculated based on the following system:
- Two assignments: 6 points
- First term exam: 17 points
- Second term exam: 17 points
- Final exam: 60 points

6 Resource Material

6.1 Purple Dragon Book (Aho, Lam, Sethi, Ullman, “Compilers principles, techniques and tools”, 2007 edition).

6.2 Red Dragon Book (Aho, Sethi, Ullman, “Compilers principles, techniques and tools”, 1986 edition).

6.3 Appel's Tiger Book (Andrew W. Appel, “Modern Compiler Implementation in C”, 1998 edition).