# INF225 Introduction into Compiler Construction, Fall 2004

## — Pensum —

**Book** pages 1–18, 31–74, 81–133, 143–217, 257–334, 345–443 **and all the Exercise Sets.**

## — Concepts and Results —

## Chapter 1 - Introduction

- Why compilers?
- Describe the translation process.
- Discuss the major data structures in a compiler.

## Chapter 2 - Scanner

- Do you know a synonym for "scanning"?

### 2.1 The Scanning Process

- What is the task of a scanner?
- What is a token, and what is the difference between a token and a string value (lexeme)?

### 2.2 Regular Expressions

- What are regular expressions good for?
- Basic operations: choice, concatenation, repetition, names
- Define the concept of regular expressions and give examples.
- Discuss useful extensions to regular expressions

## 2.3 Finite Automata

- Why do we need finite automata in addition to regular expressions?
- definition of deterministic finite automata (DFA) with examples
- Why we have to consider nondeterminism?
- Define the concept of nondeterministic finite automata (NFA) and give examples.
- From regular expressions to DFA's: Thompson's construction – subset construction – minimizing the number of states
- What is "Lex"? Give a short description.

# Chapter 3 - Context Free Grammars and Parsing

## 3.1 The Parsing Process

- What is the "structure" of a program?
- What is the task of a parser?

## 3.2 Context-Free Grammars

- definition of context-free grammars with examples
- Why it is called "context-free"? What would it mean to be "context-sensitive"?
- What is a derivation? What means leftmost and rightmost, respectively?
- What is the language defined by a grammar? How many languages are actually defined by a grammar?

## 3.3 Parse Trees and Abstract Syntax Trees

- What is a parse tree? Give an example.
- Why we use abstract syntax trees? Give an example.
- What is the difference to parse trees?
- What means "abstract"?

### 3.4 Ambiguity

- What is an ambiguous grammar?
- What are the two basic methoids to deal with ambiguity?
- How to handle precedence of operations and the dangling-else-problem?

### 3.5 Extended Notations

- Describe some useful extension of BNF.
- What is a syntax diagram?

## Chapter 4 - Top-Down Parsing

- What are backtracking parsers and predictive parsers?
- Why we don't consider backtracking parsers?

### 4.1 Top-Down Parsing by Recursive-Descent

- What is the idea of recursive-descent parsing?
- Why a grammar should always be translated into EBNF if recursive-descent is to be used?
- Is there a stack in recursive-descent parsing?
- What was your experience in writing a recursive-descent parser?

### 4.2 LL(1) Parsing

- Explain the shorthand "LL(1)".
- Explain the basic data structures and actions of LL(1) parsing with an example.
- Describe the construction of an LL(1) parsing table.
- What is "left recursion", and how to deal with it?
- Explain "left-factoring".

### 4.3 First and Follow Sets

- What are "First(X)" and "Follow(A)", and what are they good for?
- Discuss an example.

# Chapter 5 - Bottom-Up Parsing

- Explain the shorthands LR(1), LR(0), SLR(1), and LALR(1).
- What is YACC?

### 5.1 Overview of bottom-up parsing

- What is the difference between the stacks in top-down parsing and bottom-up parsing, respectively?
- Describe the two actions in bottom-up parsing.
- Why are grammars always augmented with a new start symbol?
- Explain the concepts "right sentential form", "viable prefix", and "handle".

### 5.2 LR(0) Items and LR(0) Parsing

- What is an LR(0) item?
- Describe the construction of a finite automata of LR(0) items?
- What are "closure items" and "kernel items"?
- Describe the LR(0) parsing algorithm and give an example.
- Describe the construction of an LR(1) parsing table.
- What are "shift-reduce" and "reduce-reduce" conflicts?

### 5.3 SLR(1) Parsing

- Describe the SLR(1) parsing algorithm and give an example.
- Describe the construction of an SLR(1) parsing table.

### 5.4 LR(1) and LALR(1) Parsing

- Give a short outline of LR(1) and LALR(1) parsing.
- What is the method of choice for parser generators and why?

# Chapter 6 - Semantic Analysis

- What is "semantic analysis"?

### 6.1 Attributes and Attribute Grammars

- What is an attribute?
- Describe attributes of typical pragramming language constructs.
- What is "syntax-directed semantics"?
- Define the concept of attribute grammars and give an example.

### 6.2 Algorithms for Attribute Computation

- What is described by "dependency graphs", and how we can construct them?
- What is a "topological sort"?
- Give a necessary and sufficient condition for a topological sort to exist.
- What are synthesized and inherited attributes? Give examples.

### 6.3 The Symbol Table

- What is a symbol table?
- Discuss the different possibilities to implement a dictionary structure.
- What is a "hash table"?
- How to resolve collisions?
- What is a "hash function"? Give one formula for a hash function and explain it.
- Explain the concepts "block" and "scope".
- Discuss possible symbol table structures for dealing with scopes.
- What is "sequential" and "collateral" declaration, respectively?
- Discuss the problem of recursive declarations.

### 6.4 Data Types and Type Checking

- What is a data type?
- What are "type inference" and "type checking"?
- Discuss the most common built-in types and type constructors.
- How to represent type expressions within a compiler?
- Discuss overloading, type conversion, coercion, and polymorphic typing.

## Chapter 7 - Runtime Environments

- What are the three kinds of runtime environment, and for what languages they are used?

### 7.1 Memory organization

- Describe the memory of a typical computer.
- Describe a general organization of runtime storage.
- Describe a typical procedure activation record.
- Describe the special-purpose registers program counter(pc), stack pointer(sp), frame pointer(fp), argument pointer(ap).
- What is a "calling sequence"?

### 7.2 Fully Static Runtime Environments

- What is special about fully static runtime environments?
- Give an example.
- Why do we need unnamed locations?

### 7.3 Stack-Based Runtime Environments

- When do we need stack-based runtime environments?
- Explain an example of a stack-based runtime environment.
- What are "offsets", and what they are used for?
- Discuss the corresponding calling sequence.

### 7.4 Dynamic Memory

- When do we need fully dynamic runtime environments?
- What is garbage collection?
- What is a "heap" in this context?
- Describe the implementation of a heap by a circular linked list.

### 7.5 Parameter Passing Mechanisms

- Explain the three mechanisms “pass by value”, “pass by reference”, and “pass by name”.
- Show their different behaviour by an example.

## Chapter 8 - Code Generation

- Why intermediate code?
- What is the difference between intermediate code and assembly code?

### 8.1 Intermediate Code

- Describe the three-address code and give an example.
- How three-address code is implemented?
- Describe the P-code and give an example.
- Why are there no temporaries in P-code?

### 8.2 Basic Code Generation Techniques

- How to compute code as a synthesized attribute?
- What are “macro expansion” and “static simulation”?

### 8.3 Code Generation of Data Structure References

- Discuss address calculation in three address code and in P-code
- Discuss code for array references

### 8.4 Code Generation of Control Statements and Logical Expressions

- Discuss the code generation for if- and while-statements
- How one could implement a Boolean type?
- What is a short circuit, and how it will be implemented?