Submission of assignments is optional. Submitted assignments will not be graded, but will be checked for correctness.

1 Questions to help you understand the reading

- What is the difference between a partial and a total function?
- What does it mean for a function to be computable?
- What is the halting problem?
- Give an example of a non computable function (not based on the halting problem)
- What is a grammar? How do you define it (i.e understand what is a non-terminal, terminal, productions, root)
- What is the difference between a regular, context free and context sensitive grammar?
- What is a parse tree?
- What is a derivation?
- What is an ambiguous grammar?
- How do you solve the ambiguity?
- Explain the difference between postfix, prefix and infix notation.
- What is the difference between a compiler and an interpreter?
- Explain the different phases of a compiler
- Understand the advantages/disadvantages of functional versus imperative programming
- What is the Von Neumann bottleneck?
- Understand the task of each of the following:
    1. Lexical analyser
    2. Syntax analyser
    3. Semantic analyser
    4. Intermediate code generator
    5. Code optimizer
    6. Machine code generator

- Which of the following are advantages of compilers? Which are advantages of interpreters?

  1. Generated code can run many times
  2. Can afford heavy weight optimizations
  3. Can pre-examine input program for semantic (type) errors
  4. Have full knowledge of both program input and program implementation
  5. Flexible, can easily adapt program behavior dynamically

2 Language Generated by Grammar

1. Consider the grammar G :

   $\langle S \rangle$ ::= aS | T

   $\langle T \rangle$ ::= bT | U

   $\langle U \rangle$ ::= cU | ϵ

   (a) Which of the following strings is generated by the G grammar?
   - i. ccc
   - ii. aba
   - iii. bab
   - iv. ca

   (b) Which of the following is a derivation of the string bbc?
   - i. $S \to T \to U \to bU \to bbU \to bbU \to bbc$
   - ii. $S \to bT \to bbT \to bbU \to bbcU \to bbc$
   - iii. $S \to T \to bT \to bbT \to bbU \to bbcU \to bbc$
   - iv. $S \to T \to bT \to bTbT \to bbT \to bbcU \to bbc$

   (c) Which of the following regular expressions accepts the same language as the grammar G?
   - i. $(a|b|c)^*$
   - ii. abc^*
   - iii. $a^*b^*c^*$
   - iv. $(a|ab|abc)$

2. Which of the following grammars describes the same language as 0^n1^m where $m \leq n$?

(a) $S ::= 0S1 \mid \epsilon$

(b) $S ::= 0S1 \mid S1 \mid \epsilon$

(c) $S ::= 0S1 \mid 0S \mid \epsilon$

(d) $S ::= SS \mid 0 \mid 1 \mid \epsilon$

3. What language does this grammar generate?

   $\langle S \rangle$ ::= aSa | aBa

   $\langle B \rangle$ ::= bB | b

4. What language does this grammar generate?

   $\langle S \rangle$ ::= abScB | ϵ

   $\langle B \rangle$ ::= bB | b

3 Designing Grammars

Follows some hints to generate grammars.
1. Use recursive productions to generate an arbitrary number of symbols.
2. To generate languages with matching, balanced, or related numbers of symbols, write productions which generate strings form the middle.
3. For a language that is the union of other languages, use separate nonterminals for each part of the union and then combine.

1. Design a grammar that generates zero or more a's
2. Design a grammar that generates one or more b's
3. Design a grammar that generates the language a^*b^*
4. Design a grammar that generates the language $\{a^n b^n | n \geq 0\}$
5. Design a grammar that generates the language $\{a^n b^{2n} | n \geq 0\}$
6. Design a grammar that generates the language $\{a^n b^m | 0 \leq n \leq m \leq 2n\}$
7. Design a grammar that generates words formed from $\{0, 1\}$ that begin and end with the same symbol.

8. Design a grammar that generates the language $\{a^n(b^m \mid c^m) \mid m > n \geq 0\}$
   (**Note** that this can be rewritten as $\{a^n b^m | m > n \geq 0\} \cup \{a^n c^m | m > n \geq 0\}$)

9. Design a grammar for the language L which is over the alphabet $\{a, b, c\}$, which consists of at least three consecutive b's (For example, L includes the strings *bbb* and *abcbbba*, but not *abbab*).

10. For each of the above grammars, state if the grammar is regular or context-free

4 Associativity and Priority

Rewrite the ambiguous grammar:

$$E ::= E + E \mid E * E \mid (E) \mid a \mid b$$

1. To reflect the fact that $+$ and $*$ are left associative and $*$ has higher precedence than $+$

2. To reflect the fact that $+$ and $*$ are right associative and $*$ has higher precedence than $+$

5 Parse Trees and Ambiguity

1. Show that the following grammar is ambiguous:
   $S ::= SS \mid () \mid (S)$

2. Which of the following grammars is ambiguous?

   (a) $S ::= 0SS1 \mid 0S1 \mid \epsilon$

   (b) $S ::= A1S1A \mid \epsilon$
   $A ::= 0$

   (c) $S ::= (S, S, S) \mid 1$

   (d) None of the above

3. Consider the following grammar:

⟨*assignment*⟩ ::= '`a = 1`' | '`a = 2`'

⟨*stmt*⟩ ::= ⟨*assignment*⟩ | ⟨*if-stmt*⟩

⟨*expr*⟩ ::= '`x > y`' | '`x < z`'

⟨*if-stmt*⟩ ::= '`if`' (⟨*expr*⟩) ⟨*stmt*⟩ | '`if`' '`(`'⟨*expr*⟩ '`)`' ⟨*stmt*⟩ '`else`' ⟨*stmt*⟩

Note that <> are used to denote nonterminals

Draw two parse trees for the following program fragment:

if $(x > y)$ if $(x < z)$ a = 1 else a = 2

4. Consider the following grammar for arithmetic expressions:

⟨*E*⟩ ::= E + T | E - T | T

⟨*T*⟩ ::= T * T | T / F | F

⟨*F*⟩ ::= 1 | 2 | 3 | 4 | (E)

5. Draw a parse tree for the following strings:
   (a) $1 * 2 - (3/4)$
   (b) $1 - 2 + 3$
   (c) $1 * 2 - 3/4$

6. Draw an abstract syntax tree for the following strings:
   (a) $1 * 2 - (3/4)$
   (b) $1 - 2 + 3$
   (c) $1 * 2 - 3/4$

6 Operational Semantics

This problem asks you about operational semantics for a simple language with assignment and addition. The expressions of this language are given by the grammar

$$e ::= n \mid x \mid x := e \mid e + e$$

where n can be any number $(0, 1, 2, 3, \ldots)$. We can define the operational semantics with respect to a function $\sigma : \mathit{Var} \to \mathcal{N}$, where Var is the set of variables that may appear

in expressions and $\mathcal{N}$ is the set of numbers that can be values of variables. To have some reasonable terminology, we will call σ a store and a pair e, σ a state.

When a program executes, we hope to reach a final state e, σ where e is a number, which we will call a value.

1. Let's look at arithmetic expressions. Two rules for evaluating summands of a sum are

$$\frac{e_1, \sigma \longrightarrow e_1', \sigma'}{e_1 + e_2, \sigma \longrightarrow e_1' + e_2, \sigma'}$$

$$\frac{e_2, \sigma \longrightarrow e_2', \sigma'}{n + e_2, \sigma \longrightarrow n + e_2', \sigma'}$$

   We assume that it is a single execution step to add two numbers, so we have the rule

$$\frac{n, m, p \text{ are numbers with } n + m = p}{n + m, \sigma \longrightarrow p, \sigma}$$

   The value of a variable depends on the store, as specified by this evaluation rule

$$x, \sigma \longrightarrow \sigma(x), \sigma$$

   Show how these rules let you evaluate an expression $x+y$ to a sum of numbers. Assume σ is a store with $\sigma(x) = 2$ and $\sigma(y) = 3$. Write your answer as an execution sequence of the form below, with an explanation.

$$x + y, \sigma \longrightarrow \square + \square, \sigma \longrightarrow \square + \square, \sigma \longrightarrow \square, \sigma$$

2. *Put* is the function on stores with $Put(\sigma, x, n) = \sigma'$ with $\sigma'(x) = n$ and $\sigma'(y) = \sigma(y)$ for all variables y other than x. Using *Put*, the execution rule for assignment can be written

$$x := n, \sigma \longrightarrow n, Put(\sigma, x, n)$$

   In this particular language, an assignment changes the store, as usual. In addition, as you can see from the operational semantics of assignment, an assignment is an expression whose value is the value assigned.

   If we have an assignment with an expression that has not been evaluated to a number, then we can use this evaluation rule:

$$\frac{e, \sigma \longrightarrow e', \sigma'}{x := e, \sigma \longrightarrow x := e', \sigma'}$$

   Combining these rules with the rules from part a, show how to execute $x := x + 3, \sigma$ when σ is a store with $\sigma(x) = 1$. Write your answer as an execution sequence of the form below, with an explanation.

$$x := x + 3, \sigma \longrightarrow x := \square + \square, \sigma \longrightarrow x := \square, \sigma \longrightarrow \square, \square$$

3. Show how to execute $(x := 3) + x, \sigma$ in the same level of detail, including an explanation.

4. Show how to execute $x := (x := x + 3) + (x := x + 5), \sigma$ when σ is a store with $\sigma(x) = 1$ in the same level of detail, including an explanation.