

Homework 01

Answer the questions below, and submit electronically via elearning. Make sure you submit a couple hours early at the latest and double check your submission to ensure everything is in order *before* the submission time. Your answers should be submitted as a “.doc”, “.docx”, or “.pdf” file. Your answers should be typed, but scans or pictures of hand drawn figures and diagrams may be included in the file when needed.

Due: Friday, September 24th 11:59pm

Chapter 1: Preliminaries

1. **(2pts)** What is the first successful high-level programming language designed for business use?
2. **(2pts)** Which language was most used for AI applications prior to 1990?
3. **(2pts)** Why does making a language simpler tend to make it more readable?
4. **(2pts)** Why is too much orthogonality a detriment to writability?
5. **(2pts)** What does a linker do?
6. **(2pts)** Briefly describe the three general methods of implementing a programming language.
7. **(2pts)** What role does the symbol table play in a compiler?
8. **(2pts)** What is type checking?
9. **(2pts)** What is an example of two language design criteria that are in direct conflict with each other?
10. **(2pts)** What is portability of a language?
11. **(10 pts)** Select a programming language you are familiar with such as Java or Python.
 - (a) For each criteria from chapter 1, give a score from -2 to 2. Where -2 means very bad and counter productive, and 2 means very good, enhancing the language
 - (b) Provide some code written in the language. It could be code you have written previously or even code you found on the internet. No matter the source, provide a reference (including URL when available) to give proper credit. Now, justify your scores using this piece of code as an example. You may provide more than one piece of code if that helps your arguments.

Chapter 3: Describing Syntax and Semantics

1. (2pts) Briefly describe what a lexeme and a token is.
2. (2pts) When can a parse tree be called “fully attributed”?
3. (2pts) Briefly describe the two levels of uses of operational semantics.
4. (2pts) In denotational semantics, what are the syntactic and semantic domains?
5. (2pts) What is the weakest precondition?
6. (10pts) Consider the following grammar:

$$\begin{aligned}
 \langle \text{prod_file} \rangle &\rightarrow \langle \text{prod_list} \rangle \\
 \langle \text{prod_list} \rangle &\rightarrow \langle \text{prod} \rangle \mid \langle \text{prod} \rangle \text{ ENDLINE } \langle \text{prod_list} \rangle \\
 \langle \text{prod} \rangle &\rightarrow \text{ID } \text{“-”} \langle \text{body} \rangle \\
 \langle \text{body} \rangle &\rightarrow \langle \text{body_item} \rangle \mid \langle \text{body_item} \rangle \text{ “,”} \langle \text{body} \rangle \\
 \langle \text{body_item} \rangle &\rightarrow \text{ID} \mid \text{“\&”} \mid \text{“\%”} \mid \text{“*”}
 \end{aligned}$$

Assume $\langle \text{prod_file} \rangle$ is the start symbol. Assume that all caps represents a token, quoted sequence of symbols lexemes, and angle brackets nonterminals. Also assume that the **ENDLINE** token is the groups of lexemes representing an end of line. The **ID** token represents ids in the “C” sense (start with letter or underscore, a sequence of alphanumeric and underscore characters).

- (a) Give a sentence in this language that is generated from at least 3 $\langle \text{prod} \rangle$.
 - (b) Show the left-most derivation for your sentence.
 - (c) Show the right-most derivation for your sentence.
 - (d) Show the parse tree for your sentence.
7. (10pts) Consider the following grammar:

$$\begin{aligned}
 \langle S \rangle &\rightarrow \langle As \rangle \langle Bs \rangle \mid \langle AB \rangle \\
 \langle As \rangle &\rightarrow \text{“a”} \mid \text{“a”} \langle As \rangle \\
 \langle Bs \rangle &\rightarrow \text{“b”} \mid \text{“b”} \langle Bs \rangle \\
 \langle AB \rangle &\rightarrow \text{“a”} \text{ “b”} \mid \text{“a”} \text{ “b”} \langle AB \rangle
 \end{aligned}$$

- (a) Show that this grammar is ambiguous.
- (b) Rewrite the grammar to one that is unambiguous. The new grammar must generate the exact same language. HINT: This can be done by changing a single rule.

8. (10pts) Recall the following grammar from question 6:

$\langle \text{prod_file} \rangle \rightarrow \langle \text{prod_list} \rangle$ (1)

$\langle \text{prod_list} \rangle \rightarrow \langle \text{prod} \rangle$ (2)

$\quad \quad \quad | \quad \langle \text{prod} \rangle \text{ ENDLINE } \langle \text{prod_list} \rangle$ (3)

$\langle \text{prod} \rangle \rightarrow \text{ID } \text{"-"} \langle \text{body} \rangle$ (4)

$\langle \text{body} \rangle \rightarrow \langle \text{body_item} \rangle$ (5)

$\quad \quad \quad | \quad \langle \text{body_item} \rangle \text{" , " } \langle \text{body} \rangle$ (6)

$\langle \text{body_item} \rangle \rightarrow \text{ID}$ (7)

$\quad \quad \quad | \quad \text{"\&"}$ (8)

$\quad \quad \quad | \quad \text{"\%"}^0$ (9)

$\quad \quad \quad | \quad \text{"*"}^0$ (10)

Consider the following semantic functions/predicates:

(1) Predicate: $\text{"main"} \in \langle \text{prod_list} \rangle .ids$

(2) $\langle \text{prod_list} \rangle .ids = \{ \langle \text{prod} \rangle .id \}$

(3) $\langle \text{prod_list} \rangle [0].ids = \{ \langle \text{prod} \rangle .id \} \cup \langle \text{prod_list} \rangle [1].ids$

(4) $\langle \text{prod} \rangle .id = \text{ID} .id$

$\langle \text{body} \rangle .id = \text{ID} .id$

(5) $\langle \text{body_item} \rangle .id = \langle \text{body} \rangle .id$

(6) $\langle \text{body_item} \rangle .id = \langle \text{body} \rangle [0].id$

$\langle \text{body} \rangle [1].id = \langle \text{body} \rangle [0].id$

(7) Predicate: $\langle \text{body_item} \rangle .id \neq \text{ID} .id$

This attributed grammar specifies two static semantic “rules” (restrictions on the programs in this language). What are they? Give a brief description on how they are enforced. For each rule, are the attributes involved synthesized, inherited, or a mixture of both?

9. (10pts) Recall the attributed grammar from question 8. Consider the following operational semantics:

$\langle \text{prod_file} \rangle \rightsquigarrow$	int x = 0 label ret_label = !start goto main Expand $\langle \text{prod_file} \rangle$!start: display x
$\langle \text{prod_list} \rangle \rightsquigarrow$	Expand $\langle \text{prod_list} \rangle$
$\langle \text{prod} \rangle \rightsquigarrow$	Expand $\langle \text{prod} \rangle$
$\langle \text{prod} \rangle \text{ ENDLINE } \langle \text{prod_list} \rangle \rightsquigarrow$	Expand $\langle \text{prod} \rangle$ Expand $\langle \text{prod_list} \rangle$
$\text{ID } \text{"->"} \langle \text{body} \rangle \rightsquigarrow$	ID: Expand $\langle \text{body} \rangle$ goto ret_label
$\langle \text{body_item} \rangle \rightsquigarrow$	Expand $\langle \text{body_item} \rangle$
$\langle \text{body_item} \rangle \text{" , " } \langle \text{body} \rangle \rightsquigarrow$	Expand $\langle \text{body_item} \rangle$ Expand $\langle \text{body} \rangle$
$\langle \text{body_item} \rangle \rightsquigarrow$	Expand $\langle \text{body_item} \rangle$
$\text{ID } \rightsquigarrow$	label oldret = ret_label label cur = unique_label() label ret_label = cur goto ID cur: label ret_label = oldret
$\text{"\&"} \rightsquigarrow$	int x = x + 1
$\text{"\%"} \rightsquigarrow$	int x = x*2
$\text{"*"} \rightsquigarrow$	display x

NOTE: The format is similar to our example from class. Labels in my intermediate language start with a “!”. Any without an “!” are variables that contain the label. The semantics in this questions is *different* from the one presented in other questions.

For the following programs, what will be printed when it is ran? Briefly explain why.

- (a) `main -> step,show`
 `step -> &,&`
 `show -> %,&,*`
- (b) `main -> one,two,three,four`
 `one -> two,&,*`
 `two -> &,&`
 `three -> four,*,&`
 `four -> *,*,&,%`

10. (10pts) Recall the attributed grammar from question 8. Consider the following denotational semantics:

$$\begin{aligned}
 \text{densem}(\langle \text{prod} \rangle \text{ ENDLINE } \langle \text{prod_list} \rangle) &= \text{densem}(\langle \text{prod} \rangle) \cup \text{densem}(\langle \text{prod_list} \rangle) \\
 \text{densem}(\text{ID } \text{“-”} \langle \text{body} \rangle) &= \{(\text{ID}, \text{bodysem}(\langle \text{body} \rangle))\} \\
 \text{bodysem}(\langle \text{body_item} \rangle \text{ “,” } \langle \text{body} \rangle) &= \text{bodysem}(\langle \text{body_item} \rangle) + \text{bodysem}(\langle \text{body} \rangle) \\
 \text{bodysem}(\text{ID}) &= \text{if ID} = \text{main then } 10 \text{ else } 1 \\
 \text{bodysem}(\text{“\&”}) &= 2 \\
 \text{bodysem}(\text{“\%”}) &= 100 \\
 \text{bodysem}(\text{“*”}) &= -10
 \end{aligned}$$

NOTE: The format is similar to our example from class. The semantics in this questions is *different* from the one presented in other questions.

For the following programs, what is the mathematical meaning? Briefly explain why.

- (a) `main -> step,show`
 `step -> &,&`
 `show -> %,&,*`
- (b) `main -> one,two,three,four`
 `one -> two,&,*`
 `two -> &,&`
 `three -> four,*,&`
 `four -> *,*,&,%`

11. (10pts) Recall the attributed grammar from question 8. Consider the following axiomatic semantics:

$$\begin{array}{c}
 \frac{\{Q_{\text{ID} \rightarrow \theta}, \delta = 0\} < \text{body} > \{Q_{\text{ID} \rightarrow \theta}, \delta = \theta\}}{\{Q_{\text{ID} \rightarrow \theta}\} \text{ID} \rightarrow < \text{body} > \{Q\}} \\
 \\
 \frac{\{P1\}S_1\{P2\}, \{P2\}S_2\{Q\}}{\{P1\}S_1 \text{ENDLINE } S_2\{Q\}}, \quad \frac{\{P1\}S_2\{P2\}, \{P2\}S_1\{Q\}}{\{P1\}S_1 \text{ENDLINE } S_2\{Q\}} \\
 \\
 \overline{\{Q_{\delta \rightarrow \delta+1}\} \& \{Q\}}, \quad \overline{\{Q_{\delta \rightarrow \delta+2}\} \% \{Q\}} \\
 \\
 \overline{\{Q_{\delta \rightarrow \delta*2}\} * \{Q\}}, \quad \overline{\{Q_{\delta \rightarrow \delta+ \text{ID}}\} \text{ID} \{Q\}} \\
 \\
 \frac{\{P1\}S_1\{P2\}, \{P2\}S_2\{Q\}}{\{P1\}S_1, S_2\{Q\}}
 \end{array}$$

NOTE: The semantics in this questions is *different* from the one presented in other questions.

For the following programs, assuming the precondition is {}, what is the value of **main** in the postcondition? Show the proof.

- (a) `main -> step, show`
 `step -> &, &`
 `show -> %, &, *`
- (b) `main -> one, two, three, four`
 `one -> two, &, *`
 `two -> &, &`
 `three -> four, *, &`
 `four -> *, *, &, %`