CMPSC 461, Fall 2025
Programming Language Concepts

Instructor: Dr. Suman Saha

Midterm-I Solutions

September 29, 2025 || 8:10 PM - 10:00 PM

P#	Possible	Score
1	20	
2	16	
3	10	
4	10	
5	10	
6	14	
Bonus	20	
Total	100	

This exam is a closed book and closed notes. You may not use any additional materials during the exam. All electronic devices must be put away. You may have nothing on your desk except this exam, and writing instruments. Make sure you use a pencil/pen of high contrast and your name, user ID, and student ID are clearly visible and legible in the space provided. If you fail to do so Gradescope might not process it correctly. Also please don't forget to mention your user ID on each page (in the space provided). **If you are using the extra sheet or paper to answer your question make sure you mention a note about it under that particular question.**

This exam consists of 6 questions + 1 bonus question (in total 7 questions). The bonus question is 20 marks making exam total of 100 points (80 points + 20 points for bonus questions).

How bonus question works: The maximum points you can score in this whole exam is 80 points. If you get all the first 6 six questions correct & you will score 80 points out of 100 points, you will be getting 80 marks (100 %). If you do all 7 questions correctly and score 100 out of 100 marks, you will still be getting 80 marks (100 %). If the total of your attempt in the whole exam is less than 80 points, for example: say 60 pts out of 100 pts, your final score is 60 out of 80 (75%).

Read each question carefully and use your time judiciously. The problems are not ordered by their difficulty. Also, the examples provided with the questions (if any) are not exhaustive.

Please **refer to the last page (appendix)** to refer to the rules on how to approach and solve problems related to first and follow.

User ID (Eg: abc2345): ____________________

Problem 1: Regular Expressions [20 pts]

1. Write a regular expression for a student's standard email address that satisfies the following rules: [7 pts]

   - Begins with exactly three lowercase English letters (a–z).
   - Followed by exactly four digits, where the first digit cannot be 0.
   - Followed by the symbol `@`.
   - The domain must be 3–5 lowercase letters.
   - Ends with `.edu`.

   Valid:
   `dph5402@psu.edu`
   `abc2350@math.edu`

   Invalid:
   `ab5402@psu.edu` – only 2 letters at the start
   `abc0350@psu.edu` – first digit is 0
   `abc2350@p.edu` – domain too short
   `abc2350@school.education` – must end exactly with `.edu`

2. Construct a regular expression to match U.S. ZIP codes, which are either: [6 pts]

   - Exactly 5 digits, OR
   - 5 digits followed by a hyphen and 4 more digits.

   Valid:
   `16802`
   `12345-6789`

   Invalid:
   `1234` – too short
   `123456` – too long

3. Construct a regular expression over $\Sigma = \{a, b\}$ for strings that contain an even number of `a` (0,2,4,...). [7 pts]

   Valid:
   `bb`
   `aabb`
   `babab`

   Invalid:
   `a` – odd number of `a`
   `aba` – odd number of `a`
   `aaab` – odd number of `a`

Solution

(a) `[a-z]{3}[1-9][0-9]{3}@[a-z]{3,5}\.edu`

(b) `[0-9]5(-[0-9]4)?`

(c) `b*(ab*ab*)*`

Problem 2: Finite Automata [16 pts]

1. Design a DFA recognizing the string over the alphabet $\{1, 2, 3\}$ where the string starts with 3 and has odd number of 1s. [5 pts]

   For example, accepted strings: 31, 331, 31211, 3122131; rejected strings: ϵ, 3, 311, 32211.

   Draw the graph clearly with states and transitions. Mark the start state and all accepting state(s). No explanation required. No transition table required.

2. Design a NFA recognizing the string over the alphabet $\{a, b, c\}$ where the string ends with cba. [6 pts]

   For example, accepted strings: cba, $acba$, $cccba$, $cbacba$; rejected strings: ϵ, $cbaa$, $cbba$, $cbaba$.

   Draw the graph clearly with states and transitions. Mark the start state and all accepting state(s). No explanation required. No transition table required.

3. Convert the following NFA transition table into equivalent DFA transition table using subset construction. [5 pts]

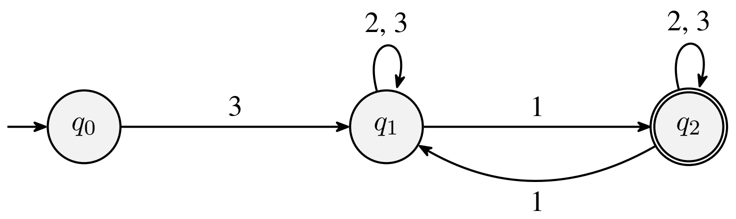
   Only DFA transition table is required. No Automata graph required. No explanation required. Also, mention the start state and all accepting state(s) in the derived DFA. You can omit the row if no transitions going out from the state or it cannot be reached from the start state.

NFA State	x	y	z
q_0	ϕ	$\{q_1\}$	$\{q_0, q_2\}$
q_1	$\{q_2\}$	ϕ	$\{q_2\}$
q_2	$\{q_0, q_1\}$	$\{q_2\}$	ϕ

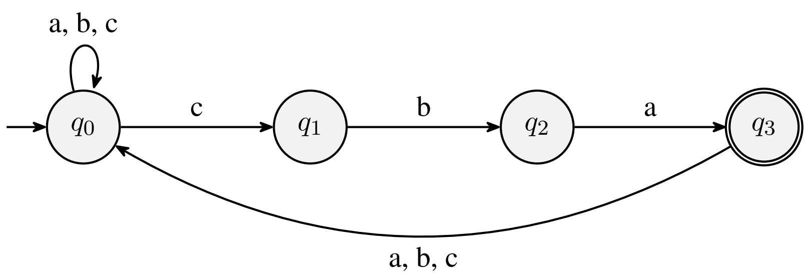
   The start state is q_0, and the accepting state is q_2.

Solution

1. One correct DFA is shown below:



2. One correct NFA is shown below:



3. The transition table is shown below:

DFA State	x	y	z
q_0	ϵ	q_1	q_0q_2
q_1	q_2	ϵ	q_2
q_0q_2	q_0q_1	q_1q_2	q_0q_2
q_2	q_0q_1	q_2	ϵ
q_0q_1	q_2	q_1	q_0q_2
q_1q_2	$q_0q_1q_2$	q_2	q_2
$q_0q_1q_2$	$q_0q_1q_2$	q_1q_2	q_0q_2

The start state is q_0, and the accepting states are q_0q_2, q_2, q_1q_2, and $q_0q_1q_2$.

——— *Extra page for Q2* ———

Problem 3: Grammar I [10 pts]
Consider the grammar:

1. $L = \{w \in \{a, b, c\}^* \mid w$ contains an even number of a's$\}$. Some valid strings are bbc, caba, aa, abba, etc.
   (a) Provide CFG for the language mentioned above. [4 pts]
   (b) Using grammar generated in part a), provide the left-most derivation for string baa. [3 pts]
2. $L = \{0^n 1^m 0^n \mid n \geq 1, m \geq 0\}$. Some valid strings in the language are 010, 00100, 00011000, etc.
   (a) Provide a CFG for the language. [3 pts]

Solution

1. (a)

$$\begin{array}{rcl} S & \rightarrow & aA \mid bS \mid cS \mid \epsilon \\ A & \rightarrow & aS \mid bA \mid cA \end{array}$$

   (b) $S \Rightarrow bS \Rightarrow baA \Rightarrow baaS \Rightarrow baa\epsilon = baa$

2.

$$\begin{array}{rcl} S & \rightarrow & 0S0 \mid 0A0 \\ A & \rightarrow & 1A \mid \epsilon \end{array}$$

Problem 4: Grammar II [10 pts]
Convert each of the following regular expressions into an equivalent CFG using BNF notation. Your grammar should be unambiguous and use a minimal number of production rules. To abbreviate ranges of consecutive characters, you can use the syntax $a \mid \cdots \mid z$.

1. `\$[1-9][0-9]?[0-9]?(,[0-9]{3})*\.[0-9][0-9]` [5 pts]

   Matches strings like `$999,888.77` and `$1,2345.67` and `$1.75` but not `$9999,888.77` or `$12,34.56` or `1.75`.

2. `(0|-?[1-9][0-9]*)(\.[0-9]*[1-9])?e(0|-?[1-9][0-9]*)` [5 pts]

   Matches strings like `0e0` and `-14.5e-5` and `1.234e10` but not `01e0` or `1e01` or `1.2000e3`

Solution

1.
$$\begin{aligned} S &\to \$\ D_1\ N_0\ N_1\ \texttt{.}\ D_0\ D_0 \\ N_0 &\to D_0\ D_0 \mid D_0 \mid \epsilon \\ N_1 &\to \texttt{,}\ D_0\ D_0\ D_0\ N_1 \mid \epsilon \\ D_0 &\to 0 \mid \cdots \mid 9 \\ D_1 &\to 1 \mid \cdots \mid 9 \end{aligned}$$

2.
$$\begin{aligned} S &\to N_0\ N_1\ \texttt{e}\ N_0 \\ N_0 &\to \texttt{0} \mid -D_1\ D_n \mid D_1\ D_n \\ N_1 &\to \texttt{.}\ D_n\ D_1 \mid \epsilon \\ D_n &\to D_0\ D_n \mid \epsilon \\ D_0 &\to \texttt{0} \mid \cdots \mid \texttt{9} \\ D_1 &\to \texttt{1} \mid \cdots \mid \texttt{9} \end{aligned}$$

——— *Extra page for Q4* ———

Problem 5: Name, Scope, and Binding I [10 pts]

Consider the following pseudo-code:

```
int val = 1;

void processor() {
    print(val);
}

void recursive_func(int n, bool shadow) {
    if (n == 0) {
        processor();
        return;
    }

    if (shadow) {
        int val = n * 10;
        recursive_func(n - 1, false);
    } else {
        recursive_func(n - 1, true);
    }
}

void main() {
    recursive_func(3, true);
}
```

(a) Draw a diagram of the **runtime stack** at the exact moment the `processor()` function is called. For each activation frame, clearly label its **static link** and **dynamic link**. [4 pts]

(b) What is the output of this program, assuming the language uses: [6 pts]

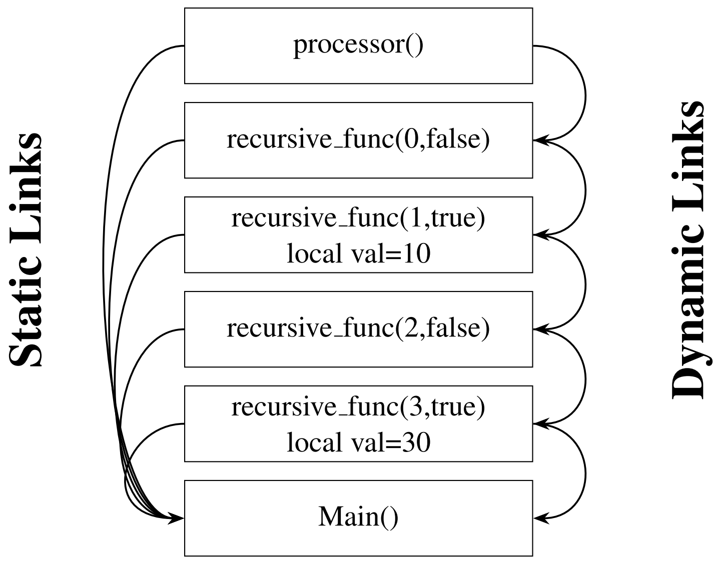
i. Static scoping?

ii. Dynamic scoping?

Solution

(a) **Runtime Stack at `processor()` call**

User ID (Eg: abc2345): ____________________



(b) **Program Outputs**

i. Static scoping: `1`

ii. Dynamic scoping: `10`

User ID (Eg: abc2345): ______________________

——— *Extra page for Q5* ———

Problem 6: Name, Scope and Binding II [14 pts]

Consider the following pseudo-code, assuming dynamic scoping rules:

```
typedef void (*FuncPtr)();

void printer() {
    print(x);
}

FuncPtr outer() {
    int x = 20;

    FuncPtr inner() {
        int x = 40;
        return printer;
    }

    return inner();
}

void main() {
    FuncPtr f = outer();
    f();
}
```

(A) Draw the **symbol tables** for the scopes `global`, `printer`, `outer`, and `inner`. [5 pts]

(B) What is the output under **shallow binding**? Explain the resolution of `x`. [5 pts]

(C) What is the output under **deep binding**? Which environment is captured and used? [4 pts]

Solution

(A) Symbol Tables

global

name	**kind**
printer	func
outer	func
main	func

printer

name	**kind**

outer

name	**kind**
x	id
inner	func

inner

name	**kind**
x	id

(B) **Shallow binding output:** Error: Unbound variable ‘x‘.

Explanation: The call ‘f()‘ (which points to ‘printer‘) happens in ‘main‘. With shallow binding, ‘x‘ is sought at the **call site**. The call stack is ‘main -> printer‘. The search checks ‘printer‘ (no ‘x‘), then its caller ‘main‘ (no ‘x‘), and then the global scope (no ‘x‘). Since no binding for ‘x‘ is found along the dynamic chain, it is an error.

(C) **Deep binding output:** `40`.

Explanation: The reference to ‘printer‘ is **created** when ‘inner()‘ returns. Deep binding captures the environment of ‘inner()‘ at that moment, which contains the binding ‘x = 40‘. When ‘printer‘ is called later from ‘main‘, it uses this captured environment (its closure) to find ‘x‘, printing ‘40‘.

——— *Extra page for Q6* ———

Problem 7: Bonus [20 pts]

1. (True or False): In a context-free grammar, the left-hand side of every production rule must consist of exactly one non-terminal symbol. [2 pts]
2. What is a production rule in a context-free grammar? Write one example and explain its parts. [2 pts]
3. Consider the grammar:

$$\begin{array}{lll} S & \rightarrow & A\,C \\ A & \rightarrow & a \mid \epsilon \\ C & \rightarrow & c\,C \mid d \end{array}$$

   (a) Compute the **FIRST** and **FOLLOW** sets for S, A, C. [7 pts]

   (b) Construct the **LL(1) parsing table** and state whether the grammar is LL(1). [9 pts]

Solution

1. True. By definition, a CFG rule has a single non-terminal on the left-hand side, while the right-hand side may contain terminals, non-terminals, or ϵ (the empty string).
2. A production rule specifies how a non-terminal can be replaced by a sequence of terminals and/or non-terminals. It has the form $A \rightarrow \alpha$, where:
   - A is a non-terminal (left-hand side).
   - α is a string of terminals, non-terminals, or ϵ (right-hand side).

   Example: $E \rightarrow E + T$
   Here, E is the non-terminal being defined, and $E + T$ shows how it can be expanded.

3. (a) FIRST sets: FIRST$(A) = \{a, \epsilon\}$, FIRST$(C) = \{c, d\}$, FIRST$(S) = \{a, c, d\}$.
   FOLLOW sets: FOLLOW$(S) = \{\$\}$, FOLLOW$(A) = \{c, d\}$, FOLLOW$(C) = \{\$\}$.

   (b) Parsing table:

Non-terminal	a	c	d	$
S	$S \rightarrow AC$	$S \rightarrow AC$	$S \rightarrow AC$	
A	$A \rightarrow a$	$A \rightarrow \epsilon$	$A \rightarrow \epsilon$	
C		$C \rightarrow cC$	$C \rightarrow d$	

   No conflicts ⇒ Grammar is LL(1).

User ID (Eg: abc2345): ____________________

——— *Extra page for Q7* ———

——— *Extra page if needed.* ———

User ID (Eg: abc2345): ____________________

——— *Extra page if needed.* ———

User ID (Eg: abc2345): ____________________

Appendix

First(C)

FIRST(C) for a grammar symbol C is the set of terminals that begin the strings derivable from C. Formally, it is defined as:

$$\text{First}(C) = \{t \mid C \to^* t\alpha\} \cup \{\varepsilon \mid C \to^* \varepsilon\}$$

Follow(C)

Follow(C) is defined to be the set of terminals that can appear immediately to the right of Non-Terminal C in some sentential form.

Rules:

- If C is the start symbol, then $ $\in$ Follow(C).
- If $C \to AB$, then First$(B) \subseteq$ Follow(A) and Follow$(C) \subseteq$ Follow(B).
- Also, if $B \to^* \varepsilon$, then Follow$(C) \subseteq$ Follow(A).

Constructing LL(1) Parsing Table

Construct a parsing table T for CFG G using the following procedure:

For each production $A \to \alpha$ in G do:

1. For each terminal $t \in$ First(α) do
   - $T[A, t] = \alpha$
2. If $\varepsilon \in$ First(α), for each $t \in$ Follow(A) do
   - $T[A, t] = \alpha$
3. If $\varepsilon \in$ First(α) and $ $\in$ Follow(A) do
   - $T[A, \$] = \alpha$