CMPSC 461, Fall 2025
Programming Language Concepts

Instructor: Dr. Suman Saha

Midterm-I Practice Exam

General Instructions about Midterm-I:

Exam Date: September 29, 2025
Exam Time: 8:10 PM - 10:00 PM

1. If you have any conflict, you are requested to contact the instructor **at least 72 hours**[1] before the exam and inform them about it.
2. If you are someone who requires any special accommodations, please reach out to the instructor **at least 72 hours** before the exam and inform them of your requirements, in order to accommodate well in advance.
3. If you are sick or unfit and are prescribed rest or quarantine by the medical authorities during the exam hours, we request you to kindly inform the instructor with the relevant paperwork.
4. Kindly arrive at the exam hall **10-15 minutes before** the scheduled time to settle yourself in and follow the instructions for seating (if any). You will have the seating plan **informed to you 36 hours** before the exam schedule.
5. It is **mandatory** to carry **Penn State Student ID** (Physical/ Mobile ID+) to the exam hall. No other ID(s) will be accepted for verifying your submission.
6. This exam will be a closed book and closed note with **no cheat sheets permitted**.
7. **No electronic devices** are allowed on the desk while taking the exam. You should only have your writing instruments and the exam paper on your desk while working on the exam.
8. Make sure your **writing instruments are high contrast** (this is required for better scanning of your submissions on Gradescope).
9. If you have any questions during the exam, kindly raise your hand and one of the proctors will reach out to you.
10. Any violations will lead to direct consequences as per the course's academic integrity policies.
11. As per Penn State policy, most campus areas no longer require masks, with a few exceptions. For safety amidst rising COVID cases, if you choose to wear a mask, we support you but remember to verify your identity (without the mask) when submitting your exam copy.

All the very best for Midterm :)

[1] unless it's an emergency- in that case, an exception will be made based on the severity of the situation.

Problem 1: Regular Expressions I

1. Construct a regular expression for URLs that:
   - Must begin with `http://` or `https://`.
   - Domain must be letters only, 2–10 in length.
   - Must end with one of the extensions: `.com`, `.org`, or `.net`.

   Valid:
   `http://psu.com`
   `https://example.org`
   `https://abc.net`

   Invalid:
   `http://psu.edu` – extension not allowed
   `https://1abc.com` – domain contains a digit

2. Construct a regular expression for time in 12-hour format `HH:MMAM` or `HH:MMPM` where:
   - `HH` ranges from 01 to 12.
   - `MM` ranges from 00 to 59.
   - The string must end with either `AM` or `PM`.

   Valid:
   `01:00AM`
   `12:45PM`
   `09:59PM`

   Invalid:
   `00:15AM` – hour cannot be 00
   `13:00PM` – hour cannot exceed 12
   `7:30AM` – hour must be two digits

3. Construct a regular expression over $\Sigma = \{a, b, c\}$ for strings where the first and last symbols are different, and the string length is at least 3.

   Valid:
   `abc`
   `bac`
   `cbbba`

   Invalid:
   `aa` – too short and same start/end
   `cb` – too short
   `cacac` – starts and ends with same symbol

Problem 2: Finite Automata I

1. Design a DFA recognizing the string over the alphabet $\{1, 2, 3\}$ where the string starts with a single 3 followed by 1 or 2.

   For example, accepted strings: 31, 32, 31231, 313323; rejected strings: ϵ, 3, 33, 123.

   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 has consecutive as.

   For example, accepted strings: *aab*, *baaac*, *aabb*, *abccaaa*; rejected strings: ϵ, *a*, *aba*, *caba*.

   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.

   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	$\{q_1, q_2\}$	$\{q_0\}$	$\{q_2\}$
q_1	$\{q_2\}$	$\{q_1\}$	ϕ
q_2	ϕ	$\{q_0, q_2\}$	$\{q_1\}$

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

Problem 3: Finite Automata II

1. Given the regular expression "$a(ab|cb)*$" over the alphabet $\Sigma = \{a, b, c\}$, build an equivalent DFA.

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

Problem 4: Context Free Grammar - I

Consider the grammar:

1. $L = \{w \in \{0,1\}^* \mid w \text{ starts with 0 and ends with 1}\}$. Some valid strings are 01, 001, 0111, 000011111, etc.
   (a) Provide CFG for the language mentioned above.
   (b) Using grammar generated in part a), provide the left-most derivation for string 0011.
2. $L = \{a^n b^{2n} \mid n \geq 1\}$. Some valid strings in the language are ab, aabbbb, aaabbbbbb, etc.
   (a) Provide a CFG for the language.

Problem 5: Context Free Grammar - II

1. (True or False): All regular grammars are also context-free grammars, but not all context-free grammars are regular. Explain your answer clearly.

2. Briefly explain the four components of a context-free grammar (CFG). Give an example production rule for illustration.

Problem 6: Ambiguity - I

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]+)*>`

   Matches strings like `<10>` and `<1234:567>` and `<12:345:6789>` but not `<01>` or `<1234:56>` or `<9876:>`

2. `(0|[1-9][0-9]*)+(0|[1-9][0-9]*)(\.(0|[0-9]*[1-9]))?`

   Matches strings like `0+0` and `123+456.7` and `0.123+456.007` but not `1.2+456` or `01+123` or `123+456.700`

Problem 7: Ambiguity - II

In Python, functions are declared using the `def` keyword, a function name, and a pair of parentheses containing a *parameter list*. This list contains zero or more variable names followed by zero or more variables with default values. This parameter list can be expressed using the following CFG:

$$
\begin{aligned}
S &\to X, Y \mid X \mid Y \mid \epsilon \\
X &\to X, X \mid \langle var \rangle \\
Y &\to Y, Y \mid \langle var \rangle = \langle expr \rangle
\end{aligned}
$$

This grammar matches inputs like $var_1, var_2 = expr_2$ but rejects inputs like $var_1 = expr_1, var_2$.

1. Show that the grammar is ambiguous using *left-most derivation* and the smallest possible example. Clearly indicate which rule is being applied at each step.
2. The *Dangling Else Problem* showed that ambiguity can change the structure of a program. Does this grammar cause the same issue? Why or why not?

Problem 8: Names, Scopes and Bindings I

Consider the following pseudo-code:

```
int x = 100;

void helper() {
    print(x);
}

void recur(int n) {
    if (n > 0) {
        int x = n;
        recur(n - 1);
    } else {
        helper();
    }
}

void caller() {
    int x = 50;
    recur(2);
}

void main() {
    caller();
}
```

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

(b) What is the output of this program, assuming the language uses:

i. Static scoping?

ii. Dynamic scoping?

Problem 9: Name, Scope, and Binding II

Consider the following pseudo-code:

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

void worker() {
    print(val);
}

FuncPtr setup() {
    int val = 50;
    return worker;
}

void recursive_executor(int n, FuncPtr F) {
    if (n > 0) {
        int val = n * 10;
        recursive_executor(n - 1, F);
    } else {
        F();
    }
}

void main() {
    FuncPtr my_func;
    my_func = setup();
    recursive_executor(2, my_func);
}
```

(A) Draw the **symbol tables** for the scopes `global`, `worker`, `setup`, and `recursive_executor`.

(B) What is the output if the language uses **shallow binding**? Justify by tracing variable resolution from the call site.

(C) What is the output if the language uses **deep binding**? Explain which referencing environment is captured when the function reference is created.

Problem 10: LL(1) Parsers

Consider the grammar:

```
S → A B
A → a A |
B → b B | c
```

(a) Compute the **FIRST** and **FOLLOW** sets for S, A, B.

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