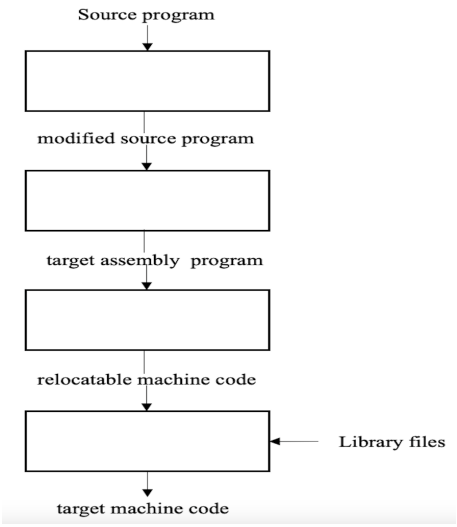
2025 NCKU CSIE Compiler Midterm Exam

Full Name : _____ Student's ID : _____

1 Compiler Process[10 pts.]

Please fill in the blanks below.(A-E) [2 pts. per blank]



A. preprocessor

B. compiler

C. assembler

D. linker/loader

E. target machine code

2 Top-down parsing [30 pts.]

Given the following grammar:

1. `START → EXPR $`
2. `EXPR → OP VAR`
3. `EXPR → VAR OP M_VAR`
4. `VAR → int`
5. `VAR → flo`
6. `VAR → (EXPR)`
7. `OP → +`
8. `OP → *`
9. `M_VAR → VAR M_VAR`
10. `M_VAR → λ`

2.1 Find first sets [10 pts.]

2.2 Find follow sets [10 pts.]

2.3 Construct LL(1) parsing table [10 pts.]

Ans:

FIRST	FOLLOW	Nonterminal
{+, *, int, flo, (}	{λ}	START
{+, *, int, flo, (}	{$,)}	EXPR
{int, flo, (}	{$, +, *, int, flo, (,)}	VAR
{+, *}	{int, flo, (, $,)}	OP
{int, flo, (, λ}	{$,)}	M_VAR

	int	flo	(	)	+	*	$
START	1	1	1		1	1	
EXPR	3	3	3		2	2	
VAR	4	5	6				
OP					7	8	
M_VAR	9	9	9	10			10

3 Regular Expression [25 pts.]

Caveats:

- Multi-line answers is **not** allowed.
- Simplify your answer as much as possible.

Hints:

- \t = `tab`
- \w = `[a-zA-Z0-9_]`
- \d = `[0-9]`

3.1 Multiple choice question [5 pts.]

Which of the following strings is **not** matched by the regular expression `"(B(AB|A)*D)*"`?

1. `BD`
2. `BABD`
3. `BAADBAABDBAD`
4. `BAADBAABBDBAD`

Ans: 4

3.2 Write a regular expression [5 pts.]

A string consisting of 3 to 7 arbitrary characters (inclusive), where the final character is a dot `"."`. Please match the **entire** string.
Examples: `89.`, `A@w@A.`, `WhoAmI.`
Ans: `^.{2,6}\.$`

3.3 Write a regular expression [5 pts.]

A string consisting of any characters **except** the lowercase letters `q`, `k`, and `v`, followed by a lowercase `b`. The length of the whole string must be even.
Examples: `(;ω;)b`, `(･ω･)b`, `o3ob`
Ans: `[^qkv]([^qkv]{2})*b`

3.4 Write a regular expression [5 pts.]

Three ways to represent a Taiwan mobile phone number starting with either `+8869` or `09`.
Examples: `+886931836217`, `0983-952-718`, `0948274395`
Ans: `(\+8869\d{8})|(09(\d{8}|\d{2}(-\d{3}){2})`

3.5 Write a regular expression [5 pts.]

A valid integer initialization in `C++`, using the following rules:

- The variable name may include letters, digits, and underscores, but cannot start with a digit.
- The assigned integer must not have leading zeros or non-digit characters.
- There is exactly one **literal space** or **literal tab** between each token.
- The statement must start with `int` and end with a semicolon `";"`.

Examples: `int a = 5;`, `int _peko_ = 777;`, `int AE86 = 6;`
Ans: `int[ \t][a-zA-Z_]\w*[ \t]=[ \t](0|[1-9]\d*);`

4 DFA / NFA

Note: The empty set should be in your minimized DFA.

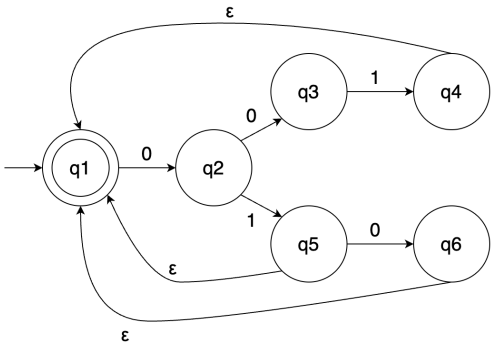
4.1 Please explain the difference between DFA and NFA in two aspects: [4 pts.]

- As for Determinism vs. Nondeterminism

- As for performance considerations (time / the number of states / construction ...)
  `Ans.`

4.1 在描述中須闡述 Determinism 提到每一個 token 都只有一種可能的 transition，Nondeterminism 則相反，有多種可能（若是僅提到其中一者扣 1%） a. 時間而言，由於 DFA 的每個 state 對每個輸入 token 的 transition 都是預先定義且只有一種可能，無需在運行時進行選擇 或回溯因此所耗時間通常比 NFA 少。 就 state 的數量和建構複雜度而言，若沒有特定前提就沒有標準答案，會依據你提出原因的合理性批改 b. NFA 有非確定性的性質（即不只一種可的 transition）和 ε-transition ，讓 transition 可以無限擴增，使得 state 數量通常比 DFA 多。但也因以上性質，對人類來說較易讀，也不需將所有 transition 畫出來，建構起來比 DFA 簡單。 c. 如果以任意 DFA 都是 NFA 來看，那麼 NFA 的 state 數量會小於等於 DFA 的 state 數量，一個 NFA 可以被轉換成一個等價的 DFA，即兩者識別相同的 RE。在轉換過程中，對原有 n 個狀態的 NFA 來說，其等價的 DFA 最多可能有 2n 個狀態。轉換後，對電腦來說較易判讀，速度也較快。 本題以其中一種面向闡述答案即可，但必須提到 DFA 的 performance > NFA 才可拿到完整分數，否則扣 1%

4.2 Transform the NFA below to a minimized DFA.[8 pts.]



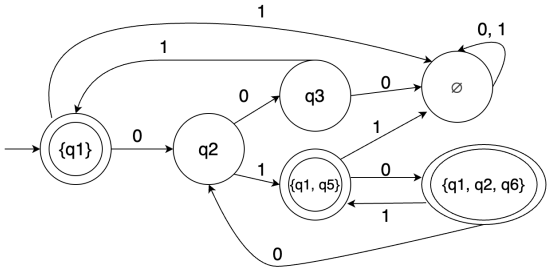
Ans. RE:$(01|001|010)^*$

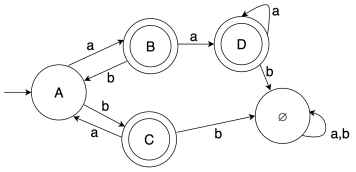
4.3 Construct a minimized DFA from `"(ab|ba)*(a+|b)"`. [8 pts.]

Ans.

5 Ambiguity [5 pts.]

Note: The upper cases are non-terminal, and the lower cases are terminal.

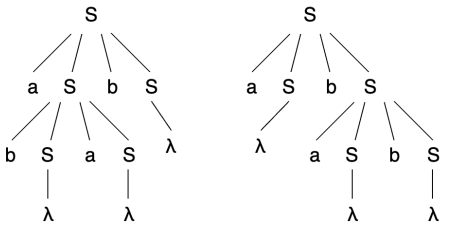




Given the grammar:

$S \rightarrow aSbS \mid bSaS \mid \lambda$

Show that this grammar is **ambiguous** by constructing two different parse trees with leftmost derivations for the sentence *abab*.
Ans.



6 Left recursion [10 pts.]

Note: The upper cases are non-terminal, and α, β are terminal.

Convert left recursion to right recursion based on the following grammar:

$A_1 \rightarrow A_1\alpha_1 \mid A_2\beta_1$
$A_2 \rightarrow A_2\beta_2 \mid A_3\beta_3 \mid A_1\alpha_2$
$A_3 \rightarrow A_2\alpha_3 \mid A_4\beta_4 \mid \alpha_4$
$A_4 \rightarrow A_3\beta_5 \mid \alpha_5$

Ans:
$A_1 \rightarrow A_2\beta_1 A_1'$
$A_1' \rightarrow \alpha_1 A_1' | \lambda$

$(A_2 \rightarrow A_2\beta_2 \mid A_3\beta_3 \mid A_2\beta_1 A_1'\alpha_2)$
$A_2 \rightarrow A_3\beta_3 A_2'$
$A_2' \rightarrow \beta_2 A_2' \mid \beta_1 A_1'\alpha_2 A_2' \mid \lambda$
$(A_3 \rightarrow A_3\beta_3 A_2'\alpha_3 \mid A_4\beta_4 \mid \alpha_4)$
$A_3 \rightarrow A_4\beta_4 A_3' \mid \alpha_4 A_3'$
$A_3' \rightarrow \beta_3 A_2'\alpha_3 A_3' \mid \lambda$
$(A_4 \rightarrow A_4\beta_4 A_3'\beta_5 \mid \alpha_4 A_3'\beta_5 \mid \alpha_5)$
$A_4 \rightarrow \alpha_4 A_3'\beta_5 A_4' \mid \alpha_5 A_4'$
$A_4' \rightarrow \beta_4 A_3'\beta_5 A_4' \mid \lambda$