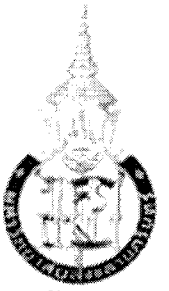


มหาวิทยาลัยสงขลานครินทร์

คณะวิศวกรรมศาสตร์

การสอบกลางภาค ประจำภาคการศึกษาที่ :1 **ประจำปีการศึกษา: 2547**

วันที่: 8 ส.ค.2547 **เวลา:** 9-12.00

วิชา: 240-340 Compiler Structures **ห้อง:** A401

ทุจริตในการสอบ โทษขั้นต่ำคือ ปรับตกในรายวิชาที่ทุจริต และพักการเรียน 1 ภาคการศึกษา

คำสั่ง: อ่านรายละเอียดของข้อสอบ และคำแนะนำให้เข้าใจก่อนเริ่มทำข้อสอบ

อนุญาต: เครื่องเขียนต่างๆ เช่น ปากกา หรือดินสอ

ไม่อนุญาต: หนังสือ, เอกสารใดๆ และเครื่องคิดเลข

เวลา: 3 ชั่วโมง (180 นาที)

ชื่อ_____รหัสนักศึกษา_____Section _____

Notational conventions:

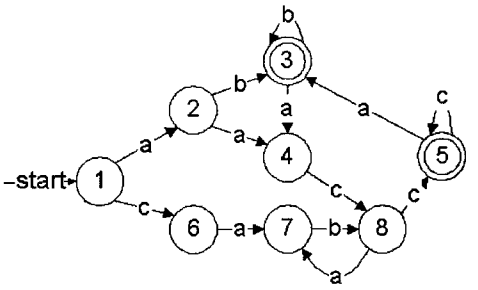
- non-terminal symbols of a grammar are in boldface Courier capital letters; example: **`TERM`**
- terminal symbols (tokens) are in boldface Courier lower-case letters, and are either given literally in double quotes, or as a token name; examples: **`"!" number "<="`**
- the symbols → and | mark off the parts of a production
- the symbol ε represents the null or empty string of zero characters

1. Consider the regular expression:

```
(0|011|0*11*)
```

over the input alphabet Σ of {0,1}. Draw the state diagram of a finite state automaton made with Thompson's Construction from this regular expression. Show each step of the construction. Be sure to show the ε transitions properly, and to indicate starting and acceptor states.

2. Consider the following finite state automaton.



a) Show the transition table for this FSA.

State	Input Symbol		
	a	b	c
1			
2			
3			
4			
5			
6			
7			
8			

b) Will the input

aacca

be accepted or rejected? Show all your work, giving the input and new state for each transition.

c) Will the input

```
abbac
```

be accepted or rejected? Show all your work, giving the input and new state for each transition.

d) Will the input

```
aacbca
```

be accepted or rejected? Show all your work, giving the input and new state for each transition.

e) Will the input

```
cababc
```

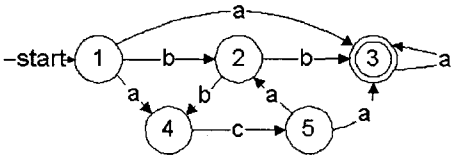
be accepted or rejected? Show all your work, giving the input and new state for each transition.

3. Show regular expressions for the following descriptions. You may use concatenation, the alternation operator |, the * operator, and parentheses.

a) An even number of zero digits, followed by exactly 3 one digits, followed by 1 or more occurrences of a zero digit and a one digit. The input alphabet Σ is {0,1}.

b) At least one binary digit; then optionally both a period character and one or more additional binary digits. The input alphabet Σ is {0,1}.

4. Why is the following finite state machine non-deterministic? List all the problems with it.



a)

b)

c)

5. Mark with "yes" or "no" whether the following productions are immediately left recursive and/or right recursive.

left recursive?	right recursive?	production
		A → A z | y
		B → w x | C x
		D → A t | B u | u v D | C
		E → A r | E r s E | ε
		F → A n | o F p | q

6. Consider the grammar:

(1) `A → A s | B C t`
(2) `B → A v | B C w | C x`
(3) `C → B y | z`

a) Eliminate all left recursion. Show your work.

b) Left factor the resulting grammar to make it suitable for predictive parsing. Show your work.

7. Describe the kind of information that goes in each of the 3 parts of a lex input file.

a)

b)

c)

8. The following tokens are used by a certain lexical analyzer:

- the single character punctuation marks: `","` `"("` `")"` `"+"` `"-"` `";"` `"="` `"{"` `"}"`
- the comparison operators: `"!="` `"=="`
- the literal reserved words: `"print"` `"while"`
- positive integer numbers consisting of one or more digital characters from `"0"` to `"9"`
- identifiers (variable names) consisting of one or more lower case English letters `"a"` through `"z"`

a) There are ordering restrictions in the above token set. For each such restriction, which token must come before which other token(s)?

b) Show the contents of a C language header file that defines symbolic names for numerical token numbers, for the above tokens. Remember that the special end-of-file token must have the value 0.

c) Show the contents of the middle (second) part of a lex input file for processing the above tokens.

9. A certain grammar using the tokens of the previous question, with productions for lists-of-statements, statements, booleans, conditions, prefix expressions and terms, is:

(1)	**L** → **S L′**	(8)	**C** → **"==" E**
(2)	**L′** → **";" S L′**	(9)	**C** → **"!=" E**
(3)	**L′** → ε	(10)	**E** → **"+" E "," T**
(4)	**S** → **id "=" E**	(11)	**E** → **"-" E "," T**
(5)	**S** → **"print" E**	(12)	**E** → **T**
(6)	**S** → **"while" B "{" L "}"**	(13)	**T** → **number**
(7)	**B** → **E C**	(14)	**T** → **id**

a) Show the parse tree with the above grammar with the input:

```
size=2; while size!=0 { print size; size=-size,1 }
```

Give a word description of each step in building the tree, with the input symbol(s) used up and the node number(s) added to the tree in each step.

b)What should the main program do to start parsing its input?

c) If the above parser is run with this input:

```
a=4; b=10; while a!=0 { print +a,b; a=-a,1; b=+10,b }
```

what output lines are printed?

d) Write pseudo-code for a production method that implements rules (2) and (3) of the above grammar, like in the lab exercise. Assume that there is a global variable called *next_token*. Show the method signature for the C language, and the processing of terminal and non-terminal symbols; if the method is not void, show how its return value is calculated. Refer to the token names in the header file, not the absolute token numbers. You may use a "2" instead of the prime character in your method name, since the prime is not legal in a C language name.

(2) **L′** → **";" S L′**

(3) **L′** → ε

e) Write pseudo-code as in the previous item, for a production method that implements rule (7) of the above grammar.

(7) **B** → **E C**

f) Write pseudo-code as in the previous item, for a production method that implements rules (8) and (9) of the above grammar (note: this method is really returning a boolean value).

(8) `C → "==" E`

(9) `C → "!=" E`

g) Write pseudo-code as in the previous item, for a production method that implements rules (10) through (12) of the above grammar.

(10) `E → "+" E "," T`

(11) `E → "-" E "," T`

(12) `E → T`