Compiler
2017 Fall Middle Examination

Name_____ Student No._____ Score_____

Problem 1: (40 points)

1

2

(a)

(b)

3

Problem 2: (60 points)

1

2

	$	=	(	)	id	pow	P	V	E
0									
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									
18									
19									
20									
21									
22									
23									
24									
25									
26									
27									
28									

3

stack	action

Problem 1: Lexical Analysis (40 points)

```
/* lex definition */

Digits                                          Digit+

Digit                                           [0-9]


%%

/* regular expressions and actions */

CHECK|SPAM|SENDER|RECEIVER|IS          {print("1"); return KEYWORD;}

Regular expression of EmailAddress     {print("2"); return ADDRESS;}

(" " | "\n")                           {print("3"); return SPACE;}

":"                                    {print("4"); return COLON;}

","                                    {print("5"); return COMMA;}

"?"                                    {print("6"); return QUESTION;}

[a-zA-Z]+                              {print("7"); return WORD;}

.                                      {error();}
```

1. The first step of lexical analysis is specifying lexical structure using regular expressions. Please write the regular expressions for EmailAddress. The requirements are as follow:(15')

   (a) EmailAddress is in the form of 'user@mail_server_name', the 'user' part and the 'mail_server_name' part with a symbol '@' linking them.

   (b) Alphabets or digits or '_' (underline) or '.' (dot) are allowed as any character of the 'user' part of the EmailAddress. But **only** alphabets or digits are allowed as the first character of the 'user' part of the EmailAddress.

   (c) The 'mail_server_name' part can be the following two types:

   First: IPv4 address in decimal form (aa.bb.cc.dd), e.g. 202.120.40.85, the four numbers in the IP address are in the range of [0,255].

   Second: Multiple (at least two) domain names linked with dot, e.g. 'dom1.dom2. … . domk', . Each domain name consists of lower or upper case letters.

Examples of EmailAddress:

Compilers@ipads.se.sjtu.edu.cn	core_19260817@Email.ADDR
515037XXXX@127.0.199.**249**	cse_ta.name@foxmail.com
This_is_valid@email.ADDR	hehe@mixedAddress.com

These are not EmailAddress:

wrong_num@1.12.123.**259**	wrong_num@1.**01.02.013**
_not_valid_user@sjtu.edu.cn	laugh@shortaddress
haha@wrong_address.COM	naughty@163.com

2. Draw the **minimized** DFA that accepts on the following requirements. For any regular expression, the minimized DFA is a unique DFA having the smallest number of states that accepts it. You will get part of scores if your DFA is not minimized. (Note that accepting state with different identities **CAN'T** be merged)(20')

   (a) Draw the minimized DFA that accepts on the number (in decimal form) in the range of [0, 255].

   (b) Draw the minimized DFA that accepts EmailAddress with the **Second** type `mail_server_name`.

3. What will be the output on the following input? Assume there is nothing to the right of the last visible character on each line **except a newline character**. The input is as follow: (5')

   CHECK

   SENDER: student@sjtu.edu.cn

   TA, I did not finish my lab, can I have my score?

   RECEIER: TA.compiler@ipads.se.sjtu.edu.cn

   IS

   SPAM

Problem 2: Grammar (60 points)

The following is a grammar for an abstracted math-related language, to calculate power and sum of variables. In this grammar, three operations are allowed: assignment, addition, power.

- Rule 1 is an assignment operation, which assigns value V to variable id.
- Rule 2 and Rule 4 are power operations.
  - Rule 2 (pow E id) means E to the power of id, i.e., E^id.
  - Rule 4 has no exponent argument, calculating the square of id, i.e., the default exponent is two.
- Rule 3 is an addition operation, calculating the sum of E and V.

NOTE: The start symbol of the grammar is 'P'. Tokens are 'Id', '=', 'pow', '(' and ')'.

P —> **id =** V	(rule 1)
V —> **pow** E **id** |	(rule 2)
E V	(rule 3)
E —> **pow (id)** |	(rule 4)
id	(rule 5)

Example:

```
To calculate 17 + 12^2 + 2^3
The program is like the following:
   result=a pow(b) pow c d
With initial values:
   a=17   b=12   c=2    d=3
```

1. Construct the state graph (DFA) for this grammar using LR(1) items. (20')
2. Follow the LR(1) procedure to construct the parsing table for the above DFA. (20')
3. Fill the table to show the operations of such a parser on the input string:

   "id=pow(id) pow pow(id) id". (20')