Exercise 1.1

- Imagine an artificial computer language, which can be utilized to solve a practical problem, i.e. the application of the language.
  - Tips 1. Language is an alternative approach to problem solving.
  - Tips 2. First find a proper problem, then design a language to solve the problem.
- Give an example of a complete piece written in the proposed language.
- Discuss how to define the new language and try your approach.
- Describe the process of changing the thinking of your language to a reality, i.e. how to make the artificial language usable.

Common Mistakes:

- Motivation is not clear: Big Integer, Tank, ...
- Only a subset of a general programming language (e.g. C/C++ or Java).
- More like Application Programming Interfaces (APIs), rather than a new computer language.
- Phrase structure of the proposed language is too simple to utilize compiling techniques, e.g. Video and Audio Management.
- A motivating example is the best way to reveal the characteristics of your language.

问题描述：

在学习与数学相关学科的内容时常常会编辑一些数学表达示，目前市场上流行的编辑器一般都是图形化的编辑工具，即所见即所得，这种编辑工具比较直观，可以直接将需要编写的符号拖拉，但是在使用此类工具编辑公式时一个显著的问题就是编辑繁琐不堪，编辑效率极其低下，因此我所做的工作就是开发一种文本式的“数学公式编辑语言 MEEL”，虽然它不会像普通编辑器那样直观，但是一旦掌握了方法之后可以大大提高我们编辑速度，提高办公效率。

MEEL 的 BNF 定义：

Var ::= letter { letter | digit}

Letter ::= A | B | C | .... | Z | a | b | c | .... | z

Digit ::= 0 | 1 | 2 | ... | 9

Data ::= [symbol] usigned [point usigned]

Symbol ::= + | -

Usigned ::= digit { digit }

Point ::= .

Base_operation ::= + | - | * | / | = | | | % | &

Bracket ::= ( ) | [ ] | { }

AddLoop_Statement $\sum$ A ::= “Add” “(“ <expression(A)> “,” ((expression(B) “ ,”) | “,”) (((expression(C) “)”) | “)”)

POW_statement A ::= “Pow” “(“ <expression(A)> “,” ((expression(B) “,”) | “,”) (((expression(B) “)”) | “)”)

Sqrt_Statement $\sqrt{A}$ ::= “Sqrt” “(“<expression(A)> (((expression(B) “)”) | “)”)

Expression_Statement ::= [symbol] (([bracket] var) | ([bracket] data)) { (base_operation Add | Pow | Sqrt | (([bracket] var) | ([bracket] data)) | (“(“[symbol] (([bracket] var) | ([bracket] data)) “)”) | expression

Instance :

已知想要得到的公式为：

Language Name: Jubilant Musician (J M)

Function: Jubilant Musician language is designed for recording music book. It can display the music book in several common forms such as staff or numbered musical notation.

Characteristic: simple , convenient & easy to learn

Syntax:

1. Source code should be written in ASCII.
2. The source code file should be saved as the form of (*. jm) For instance, LittleStar.jm
3. The only way for noting is to write notes after double slash "//"
4. On .jm file consists of three parts as follows: basic information ,score and lyric. Basic information and score are compulsory while lyric is alternative. There is no restriction of the sequence of the three parts. The three parts should follow the prefix "@INFO@", "@SCORE@" and "@LYRIC@" respectively.
5. Basic information part should contain several items as follows:

Item	Prefix	Importance	Sample
Name	@NAME@	Compulsory	@NAME@ Canon and Gigue in D
Composer	@COMPOSER@	Alternative	@COMPOSER@ Johann Pachelbel
Written Time	@WTT@	Alternative	@WTT@ 1688
Style	@STYLE@	Alternative	@STYLE@ classic
Rhythmic pattern	@RHYPATTERN@	Compulsory	@RHYPATTERN@ 4/4
Tempo	@TEMPO@	Compulsory	@TEMPO@ 80
Key	@KEY@	Compulsory	@KEY@ D
Major or minor	@MORM@	Compulsory	@MORM@ Major

Comments:

Style: could be as follows
mass ,oratorio ,passion ,symphony ,concerto ,sonata ,sonatina ,band ,march ,jazz ,waltz ,fantasia ,l ullaby ,tango ,serenade minuet,aria ,polka ,rumba ,disco ,suite ,fugue and so on

Rhythmic pattern: could be as follows
2/2 2/4 2/8 3/2 3/4 3/8 4/2 4/4 4/8 6/2 6/4 6/8 12/4 12/8 12/16 7/4 11/7
and so on

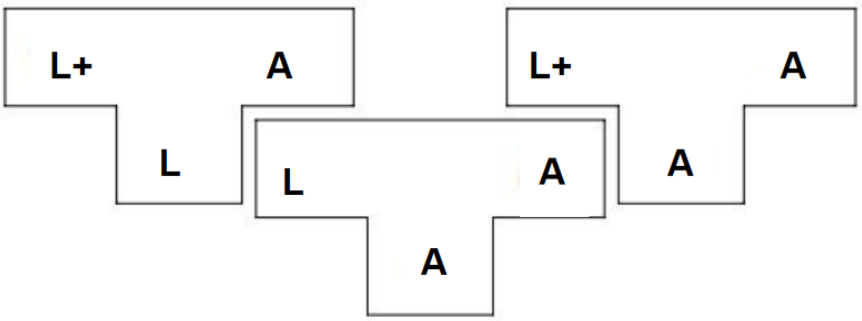
Exercise 1.2

- Draw a T-diagram with two stages of bootstrappings.
  - Given a new programming language L++, we firstly implement L, a small subset of L++.
  - Then we use L to implement L+, a subset of L++ and a superset of L.
  - Finally, L++ is implemented using L+.

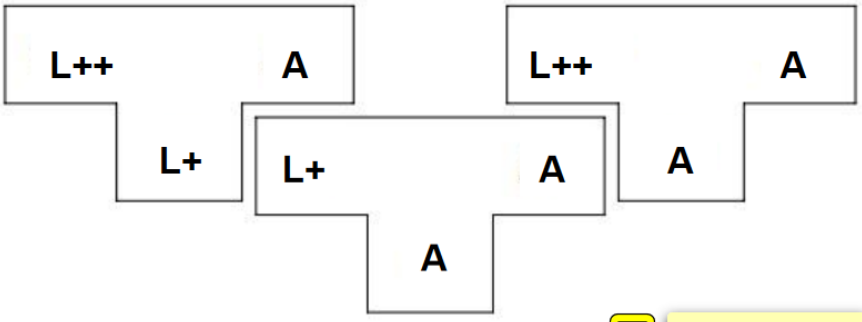
Common Mistakes:

- Two stages of bootstrapping are not merged to a single T diagram.

Stage one:



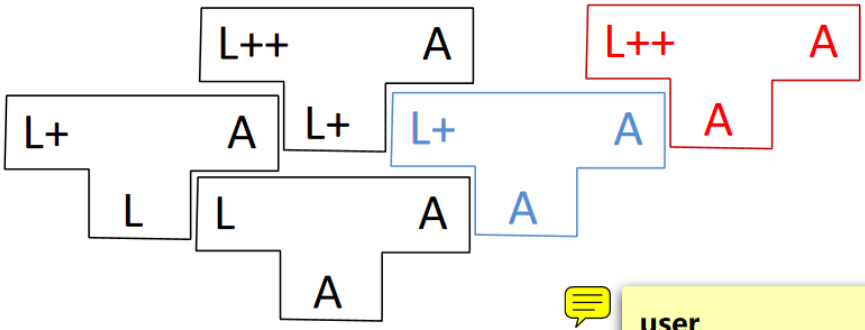
Stage two:



user

二图可合并！

Suppose machine code A could be directly ran on the host machine:



user

Leeman: Very Good !

Exercise 2.1

- Given the following grammar:

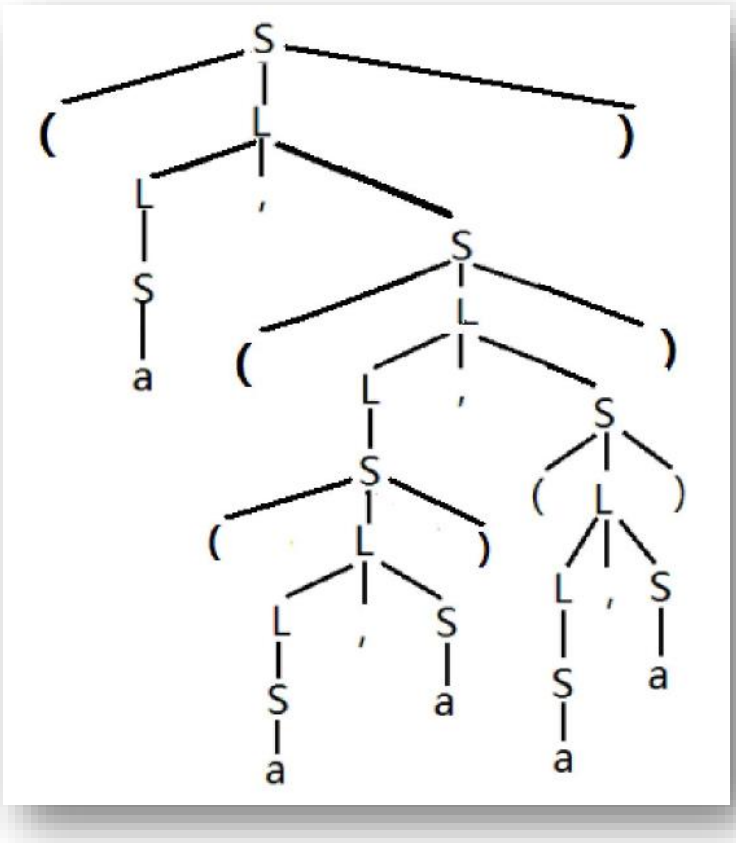
$$S \rightarrow \mathbf{(}\ L\ \mathbf{)} \mid \mathbf{a}$$

$$L \rightarrow L\ \mathbf{,}\ S \mid S$$

Construct a parse tree for the sentence
(a, ((a, a), (a, a)))

Common Mistakes:

- The root of the parse tree is not the **start symbol** of the grammar.
- Some terminals are missed in the parse tree, usually the epsilon, an operator and a parenthesis.



Exercise 2.2

- Given the following grammar:

  bexpr → bexpr **or** bterm | bterm

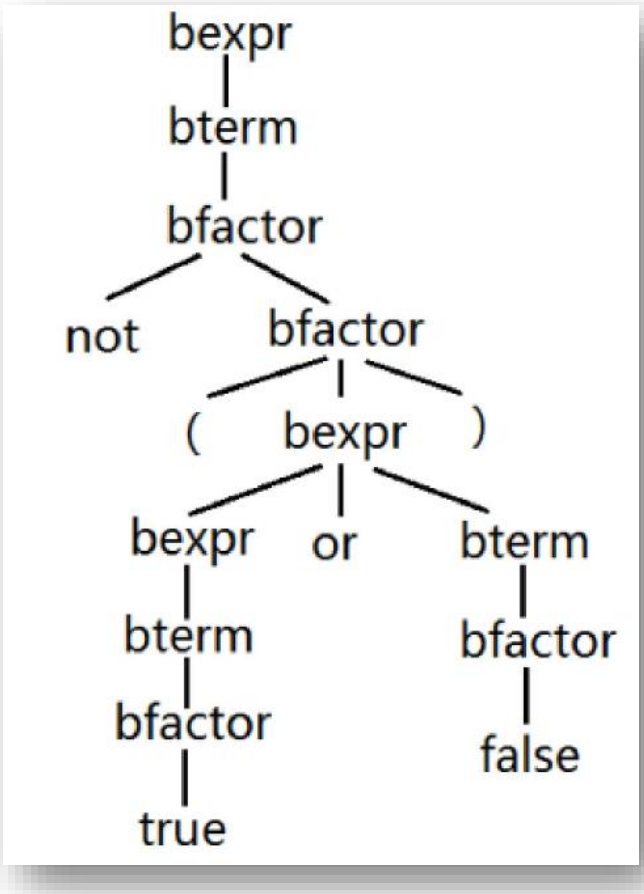
  bterm → bterm **and** bfactor | bfactor

  bfactor → **not** bfactor | **(** bexpr **)** | **true** | **false**

  Construct a parse tree for the sentence **not (true or false)**

Common Mistakes:

- Swap the position of symbols in a production, e.g: "bexpr -> bexpr or bterm" is written "bexpr -> bterm or bexpr".

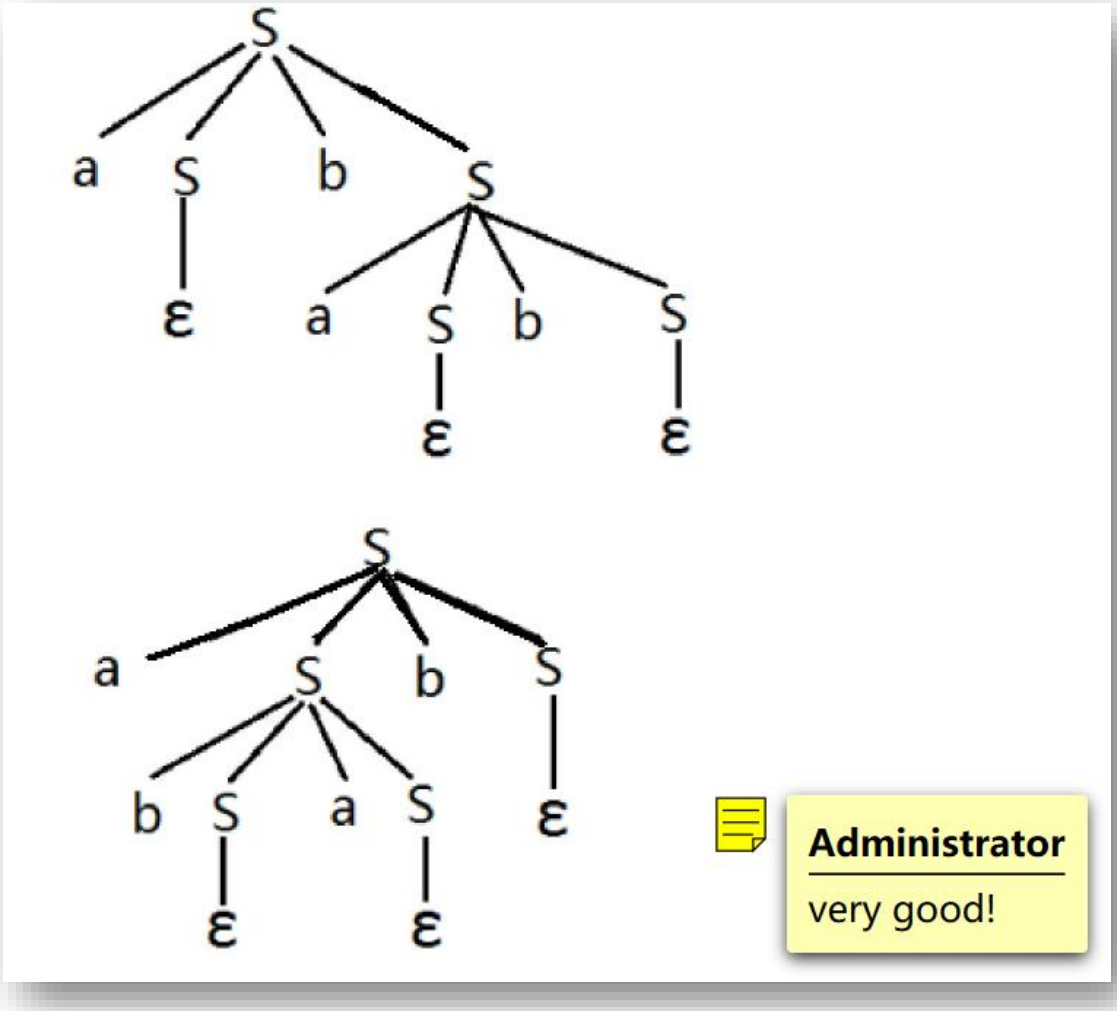


Exercise 2.3

- Is the grammar:
  S → a S b S | b S a S | ε
  ambiguous? Why?

Common Mistakes:

- Misunderstand the general approach to judge whether a given grammar is ambiguous.



Exercise 3.1

- Give the recognized tokens of the following program in Pascal.

```
function max(i, j: integer): integer;
{return the maximum of integers i and j}
begin
  if i > j then max := i else max := j
end;
```

Common Mistakes:

- Comments are considered as tokens.
- Merge operators and punctuation into one class.
- List the classification and set of tokens, but not the sequence of tokens.

词素	类型
function	Reserved words
max	Identifier
(	Punctuation
i	Identifier
,	Punctuation
j	Identifier
:	Punctuation
integer	Identifier
)	Punctuation
:	Punctuation
integer	Identifier
;	Punctuation
begin	Reserved words
if	Reserved words
i	Identifier
>	Operators
j	Identifier
then	Reserved words
max	Identifier

Exercise 3.2

- (DBv2, Ch.3, pp.125, ex.3.3.2) Describe the languages denoted by the following regular expressions:
  - a (a | b)* a
  - a* b a* b a* b a*

Common Mistakes:

- Use inappropriate partial enumeration of language a*ba*ba*ba*, e.g {bbb, abbb, ababb, abababb, abababa, aabbb, aababb, aabababb, aabababa, aabaabb, ...}

a(a|b)*a：分别以 a 开头和结尾，中间有任意个 a 或 b 组成的语言。

a* b a* b a* b a*:

In this language,a program should contain exactly three 'b',and there are arbitrary amount of 'a' between,before and after the three 'b'.

Exercise 3.3

- (DBv2, Ch.3, pp.125, ex.3.3.4) Most Languages are case sensitive, so keywords can be written only one way, and the regular expressions describing their lexemes are very simple.
- However, some languages, like Pascal and SQL, are case insensitive. For example, the SQL keyword **`SELECT`** can also be written **`select`**, **`Select`**, or **`sELEcT`**.
- Show how to write a regular expression for a keyword in a case insensitive language. Illustrate your idea by writing the expression for **`SELECT`** in SQL.

```
(S|s)(E|e)(L|l)(E|e)(C|c)(T|t)
```

Exercise 3.4

- Given the following regular expression

$$1^*(0 \mid 01)^*$$

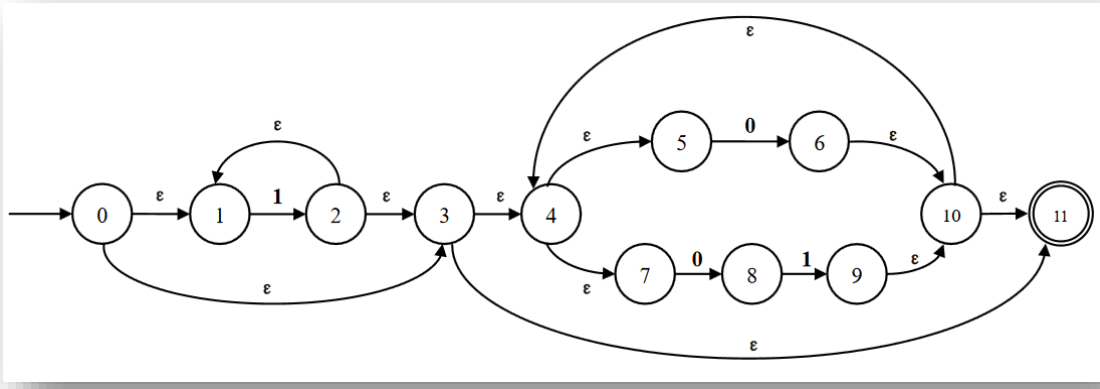
(1) Transform it to an equivalent finite automaton.

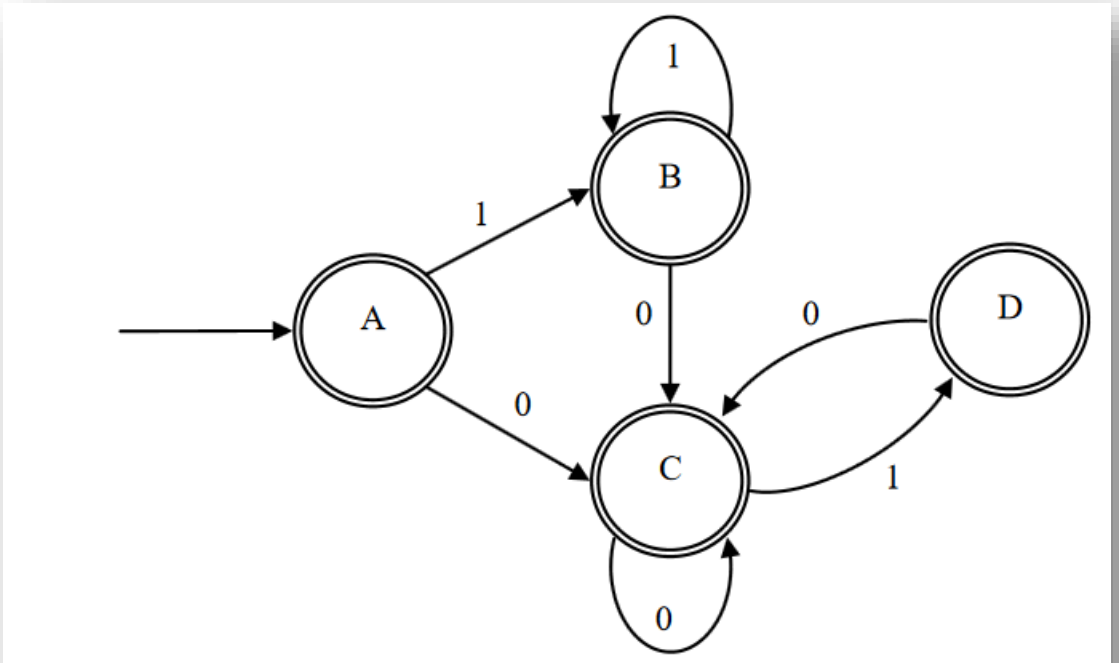
(2) Construct an equivalent DFA for the result of exercise (1).

(3) Reduce the result of (2) and get a reduced DFA.

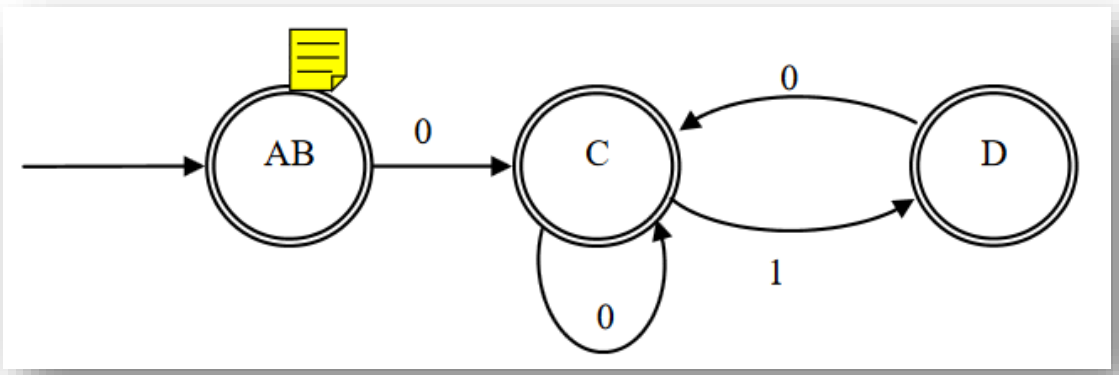
Common Mistakes:

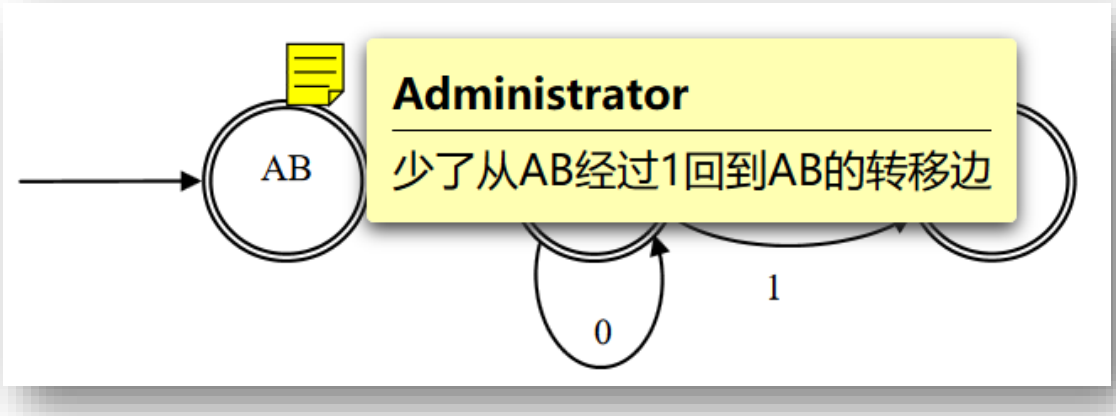
- Do not specify the corresponding NFA states of a DFA state when converting an NFA to DFA.
- Do not strictly follow the Thompson Construction algorithm of transfering a regular expression to an NFA.





其中　　A={0,1,3,4,5,7,11}
B={2,1,3,4,5,7,11}
C={6,10,11,4,5,7,8}
D={9,10,11,4,5,7}




Administrator
少了从AB经过1回到AB的转移边
AB
0
1

Exercise 3.5**

- Given the alphabet Σ = { z, o, / }, a comment in a program over Σ begins with "/o" and ends with "o/". Embedded comments are not permitted.
  (1) Draw a DFA that recognizes nothing but all the comments in the source programs.
  (2) Write a single regular expression that exactly describes all the comments in the source programs.

Common Mistakes:

- Over-specification: illegal strings are recognized in the DFA.

这是从 C 语言注释抽象出来的一个经典正则表达式问题，该问题将实际中的/* ... */语法作了简单修改，从而避免了语法中的星号与正则表达式元符号的混淆。一个指定语言的正则表达式不是惟一的，因而存在多个正确的答案。

(1) 以下是常见的 3 种有效答案：

参考答案 1：

```
/o(o*z | /)*o+/
```

参考答案 2：

```
/o/*(o*z/*)*o+/
```

参考答案 3：

```
/o(/*o*z)*/*o+/
```

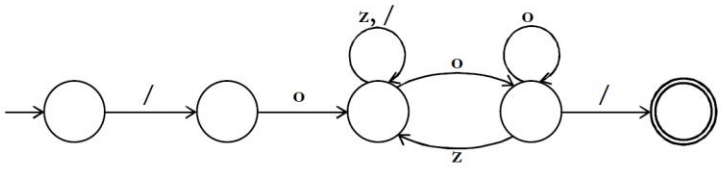
上述每一答案的关键是位于中间的配对表达式。上面的每一个配对表达式都设计为避免出现一个 o 后面紧跟一个/的情况，它们都采用 o*z 模式表示一个 z 前面有一个可选的 o 序列，这正是求解问题的关键。剩下的就是要注意让/可出现在该部分的任何地方，在仔细思考后应不难写出。

另一值得考虑之处是：由于 o*z 模式总是要求注释体中每串 o 序列后都紧跟一个 z，故需要专门处理一串 o 序列直接到注释结尾的情况。因而，上述各种答案都是以 o+/结尾而不是以 o/结尾。如果注释的模式以 o/结尾，就无法匹配/ooo/这一类合法的注释。

常见错误：

- 匹配了太多的串：显然/o(o|z|/)*o/模式会将/oo/zzzz/oo/误识别为一个正确的注释。
- 在试图避免 o/时遗漏了我们通过(...)*表达式多次重复出现的情况，例如/o(/|oz|oo)*o+/。
- 不允许注释在/o 开头后立即跟着一个或多个/。
- 不允许注释在 o/结尾前立即放着一个或多个 o。
- 不允许注释体中出现任何的 o 或/。
- 不允许注释体中出现任何的/o。
- 不支持形如/oooo...oooo/的注释。
- 将/o/接受为一个合法的注释。

(2) 其实先构造一个有限自动机，再写出上一小题的正则表达式可能是更方便的解题途径。识别上述语言的最小 DFA 如下：



解题的关键是 DFA 中两个带自环的状态，左边那个状态用于消耗(z | /)*的所有串，右边那个状态用于消耗 o*的所有串。只要有一串 o*开始，回到左边状态的惟一途径是再读入一个 z；如果此时读入的是一个/，则识别了一个完整的注释。

另一个值得注意的是终结状态是一个死状态，它不会再转入任何其他状态。这保证了 DFA 不会读入第一个 o/之后，不会再读入任何其他东西。

Exercise 4.1

- Given the following grammar

$$S \rightarrow (L) \mid a$$
$$L \rightarrow L, S \mid S$$

  - Eliminate left recursions in the grammar.
  - Draw the transition diagrams for the grammar.
  - Write a recursive descent predictive parser.
  - Indicate the procedure call sequence for an input sentence `(a, (a, a))`.

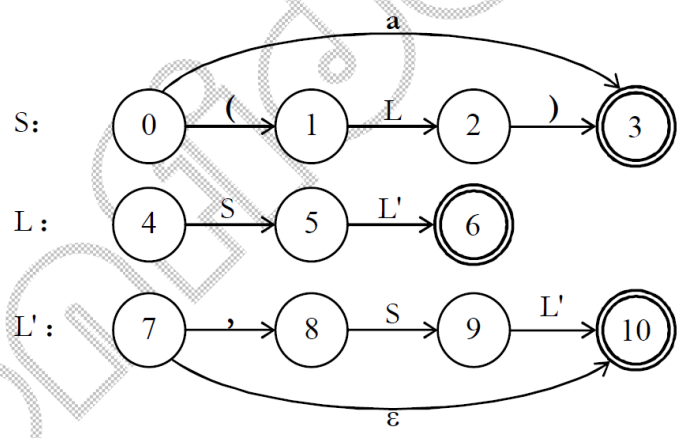
Common Mistakes:

- No error reporting actions when syntax error arises.
- Do not define the behavior of match() function.
- Redundant arrow pointing to the start state of the transition diagrams.
- Give a parse tree (or match/derive sequence) instead of procedure call sequence.

(1) 消除左递归：

$$S \rightarrow (L) \mid a$$
$$L \rightarrow SL'$$
$$L' \rightarrow ,SL' \mid \varepsilon$$

(2) 作图（注意每一个非终结符号都应有一个图，状态从 0 开始连续编号。）



(3) 预测分析器程序：

```
void match(Token t) {
        if (lookahead == t) {
                getNextToken();
        } else  error();
}
void S() {
        if (lookahead == 'a') {
                match('a');
        } else if (lookahead == '(') {
                match('('); L(); match(')');
        } else  error();
}
void L() {
        S(); L2();
}
void L2() {
        if (lookahead == ',') {
                match(','); S(); L2();
        }
}
```

常见错误：

- 当转换图中有一条ε路径时，不应在最后的 **else** 分支中调用 error()；而没有这条ε路径时，则必须在最后的 **else** 分支中调用 error()。

(4) 句子(a,(a,a))的处理过程：

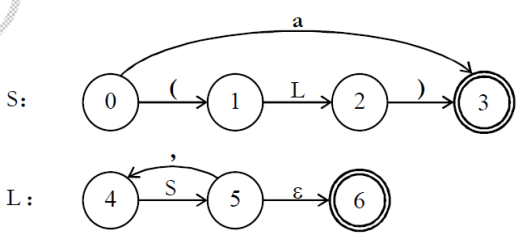
```
S()
⇒ match('(')
⇒ L()
        ⇒ S()
                ⇒ match('a')
        ⇒ L2()
                ⇒ match(',')
                ⇒ S()
                        ⇒ match('(')
                        ⇒ L()
                                ⇒ S()
                                        ⇒ match('a')
                                ⇒ L2()
                                        ⇒ match(',')
                                        ⇒ S()
                                                ⇒ match('a')
                                        ⇒ L2()
                                                ⇒
                        ⇒ match(')')
                ⇒ L2()
                        ⇒
⇒ match(')')
```

常见错误：

■ 没有用合适的方式将递归程序的分析过程展现出来，正确的做法是使用树或缩进格式等。

(2′) 另一种做法是将上述转换图优化，然后再写出对应的递归下降预测分析器源程序。可惜转换图的转换并没有什么固定的算法，只能根据特定问题进行 ad hoc 的变换，主要采用的思路是变换后代入。

将 L' 的转换图变换并代入到 L 的转换图，可得化简化的转换图如下：



(3′) 预测分析器程序：

```
void match(Token t) ...
void S() {
        if (lookahead == 'a') {
                match('a');
        } else if (lookahead == '(') {
                match('('); L(); match(')');
        } else error();
}
```

Exercise 4.2

- Consider the context-free grammar

$$S \rightarrow aSbS \mid bSaS \mid \varepsilon$$

  - Can you construct a predictive parser for the grammar? and why?

Common Mistakes:

➢ Misunderstand the conditions for predictive parsing.

不可，因为该文法是一个二义文法，而所有二义文法都无法直接采用预测分析技术。

证该文法是二义的，只需找出一个输入串（例如 abab）存在两个最右推导：

$S \Rightarrow aSbS \Rightarrow aSb \Rightarrow abSaSb \Rightarrow abSab \Rightarrow abab$

$S \Rightarrow aSbS \Rightarrow aSbaSbS \Rightarrow aSbaSb \Rightarrow aSbab \Rightarrow abab$

从二义文法角度可能是最简单、最直接的论证。当然也可以从 LL(1)文法性质的其他方面入手。

答：不能构造一个 predictive parser。对于此 CFG，

FIRST（S）={a，b，ε}，FOLLOW（S）={a，b，$}，

显然 FIRST(α)∩ FOLLOW(S) ≠ ∅ 所以无法构造一个 predictive parser

Exercise 4.3

- Compute the FIRST and FOLLOW for the start symbol of the following grammar

$$S \rightarrow SS+ \mid SS* \mid a$$

Common Mistakes:

- Missing $ in the FOLLOW set.

$$FIRST(S) = \{ a \}$$

$$FOLLOW(S) = \{ +, *, a, \$ \}$$

Exercise 5.1

- Given the following grammar

$$S \rightarrow (L) \mid a$$

$$L \rightarrow L, S \mid S$$

  - Construct an LL(1) parsing table for the grammar
    - Note: you must eliminate the left recursion first.
  - Draw the detailed process of the parsing of the sentence `(a, (a, a))`, follow the style in the previous slides.

Common Mistakes:

- Redundant '$' in FOLLOW(L') when there is a production L' -> ,SL' | eps , and S is the start symbol.

(1) 消除左递归后文法为：

S → (L) | a
L → SL'
L' → ,SL' | ε

(2) 构造 LL(1)分析表如下：

	a	,	(	)	$
S	S → a		S → (L)		
L	L → SL'		L → SL'		
L'		L' → ,SL'		L' → ε	

(3) 句子(a,(a,a))的分析过程如下：

序号	堆栈	输入串	动作	输出
1	$S	(a,(a,a))$	*derive*	S → (L)
2	$)L(	(a,(a,a))$	*match*	
3	$)L	a,(a,a))$	*derive*	L → SL'
4	$)L'S	a,(a,a))$	*derive*	S → a
5	$)L'a	a,(a,a))$	*match*	
6	$)L'	,(a,a))$	*derive*	L' → ,SL'
7	$)L'S,	,(a,a))$	*match*	
8	$)L'S	(a,a))$	*derive*	S → (L)
9	$)L')L(	(a,a))$	*match*	
10	$)L')L	a,a))$	*derive*	L → SL'
11	$)L')L'S	a,a))$	*derive*	S → a
12	$)L')L'a	a,a))$	*match*	
13	$)L')L'	,a))$	*derive*	L' → ,SL'
14	$)L')L'S,	,a))$	*match*	
15	$)L')L'S	a))$	*derive*	S → a
16	$)L')L'a	a))$	*match*	
17	$)L')L'	))$	*derive*	L' → ε
18	$)L')	))$	*match*	
19	$)L'	)$	*derive*	L' → ε
20	$)	)$	*match*	
21	$	$	*accept*	

常见错误：

- 无最后的接受动作。
- 无反序将产生式右部压入堆栈。

Exercise 5.2 **

- Given the following grammar

$$A \rightarrow B \mid B\,C$$
$$B \rightarrow \mathbf{a}\,B \mid \varepsilon$$
$$C \rightarrow \mathbf{a}\,\mathbf{b}$$

  - Left factor the grammar.
  - After left factoring, is the grammar an LL(1) grammar? or is it an LL(k) grammar? and why?
    - Note: you may try the input string **ab**.

Common Mistakes:

- Cannot explain why a grammar is not LL(1).
- Mix the definition of non-LL(1) grammar and ambiguious grammar.

5.2.(1)After the left factor:

A→BA'

A' →C|ε

B→aB|ε

C→ab

(2)When lookahead=1, FIRST and FOLLOW:

First(A)= First(A')=First(B)={a, ε}

First(C)={a}

Follow(A)=Follow(A') =Follow(C)={$}

Follow(B)={a,$}

Administrator

最好还是把b这一列画出来

The Parsing Table:

Non-terminal	lookahead	
	a	$
A	BA'	ε
A'	C	ε
B	aB or ε	ε
C	ab	

Exist more than one element in the table,so it is not an LL(1) grammar.

When lookahead=2,FIRST and FOLLOW:

First(A)={aa,ab,ε}

First(A')={ab,ε}

First(B)={aa,ε}

First(C)={ab}

Follow(A)={$}

Follow(A')={$}

Follow(B)={ab,$}

Follow(C)={$}

The Parsing Talbe:

Non-terminal	lookahead		
	aa	ab	$
A	BA'	BA'	ε
A'		C	ε
B	aB	ε	ε
C	ab		

This table is legal,so it is an LL(2) grammar.

Exercise 6.1

- Consider the grammar

$$S \rightarrow (L) \mid a$$
$$L \rightarrow L,S \mid S$$

- We have the following operator-precedence relations for the grammar.
  - Show the detailed process of the parsing of the sentence **(a, (a, a))**, follow the style in the previous slides.

	a	(	)	,	$
a			≻	≻	≻
(	≺	≺	≡	≺	
)			≻	≻	≻
,	≺	≺	≻	≻	
$	≺	≺			

Common Mistakes:

- Try to eliminate Left-Recursion with an OPP grammar.
- Misunderstand the concept of Left-Most Prime Phrases, which leads to many kinds of wrong answers.

Step	Stack	Input	Reference	Action	O
1	$	(a,(a,a))$	$ < (	shift	
2	$(	a,(a,a))$	(< a	shift	
3	$(a	,(a,a))$	a > ,	reduce	S → a
4	$(S	,(a,a))$	(< ,	shift	
5	$(S,	(a,a))$	, < (	shift	
6	$(S,(	a,a))$	(< a	shift	
7	$(S,(a	,a))$	a > ,	reduce	S → a
8	$(S,(S	,a))$	(< ,	shift	
9	$(S,(S,	a))$	, < a	shift	
10	$(S,(S,a	))$	a >)	reduce	S → a
11	$(S,(S,S	))$	, >)	reduce	L → S
12	$(S,(S,L	))$	, >)	reduce	L → L,S
13	$(S,(L	))$	(=)	shift	
14	$(S,(L)	)$	) >)	reduce	S → (L)
15	$(S,S	)$	, >)	reduce	L → S
16	$(S,L	)$	, >)	reduce	L → L,S
17	$(L	)$	(=)	shift	
18	$(L)	$	) > $	reduce	S → (L)
19	$S	$		accept	

Administrator

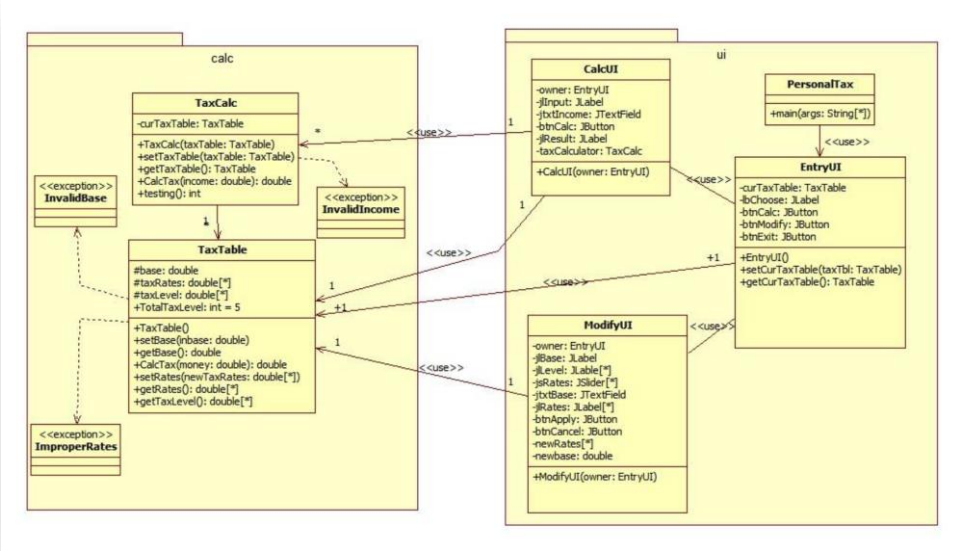
prime phrase不一定是simple phrase，可以直接将a归约成L

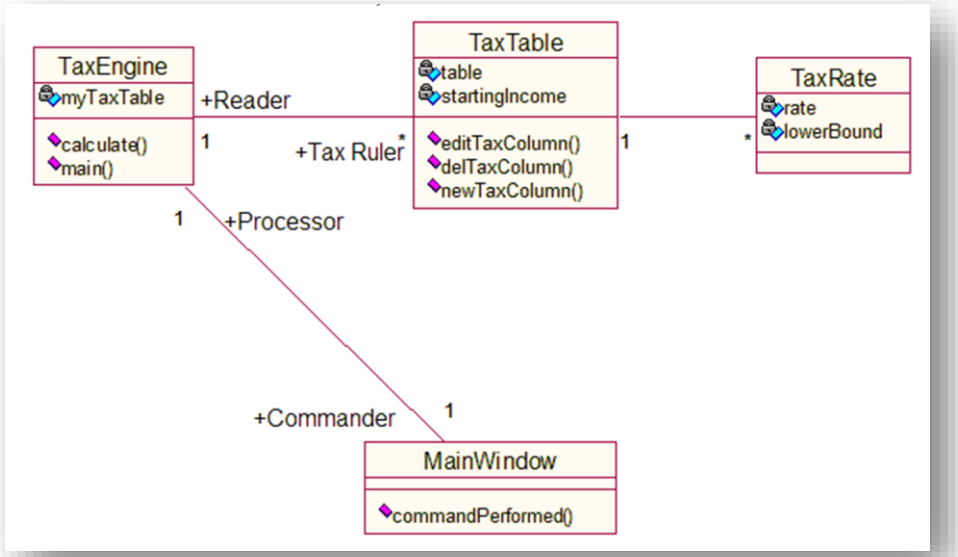
中山大学
SUN YAT-SEN UNIVERSITY

Version 3.00; Sep. 14, 2017

编译原理实验

计算个人所得税 *PersonalTax*

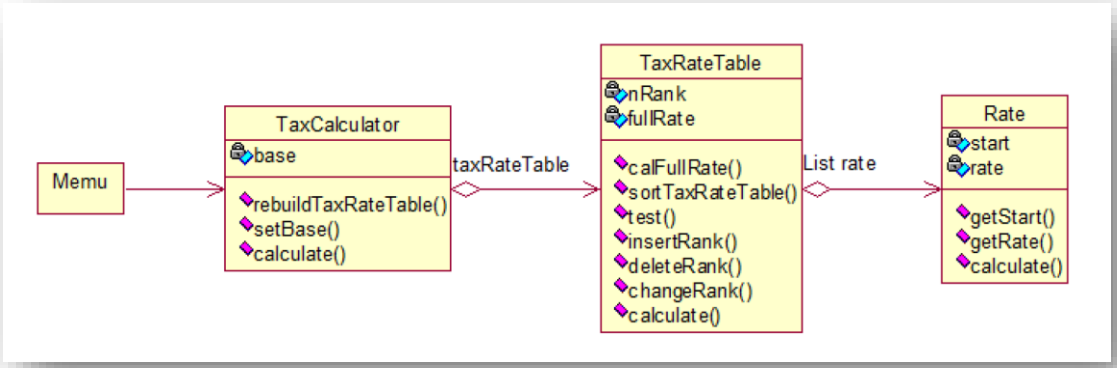




2 Class Design

To fulfill the functions of the Calculator, I have designed several classes.

- **Package personaltax**
  - **Rate:** Stand for a single rank of the tax rate table, with starting money and rate as its attribute, and a method to calculate the rate for this rank.
  - **TaxRateTable:** Contain information of all the ranks of the tax rate, using a list of class Rate's instances to form the table, with methods of insert, delete, change to make some changes to a single rank of the table, called local change, and also a method to calculate the full tax for an income.
  - **TaxCalculator:** A base of tax rate, a tax rate table and methods acting on them compose the class Calculator. The base can be changed here. Also, a total new table may be created while destroying the old one, known as global change. In the same way, a method to calculate the rate should be implemented here.
- **Package myException**
  - **NegativeException:** Thrown when the num is a negative where it is supposed to be a positive.
  - **OutOfRangeException:** Thrown when the index of rank is below zero or larger than the total num of ranks existing.
  - **TableIllegalException:** Thrown when the tax rate table is illegal. Illegal table here means that two or more ranks have the same starting money, or a higher rank's rate is not larger than a lower one.
- **Package UI**
  - **Memu:** Interface to customers.

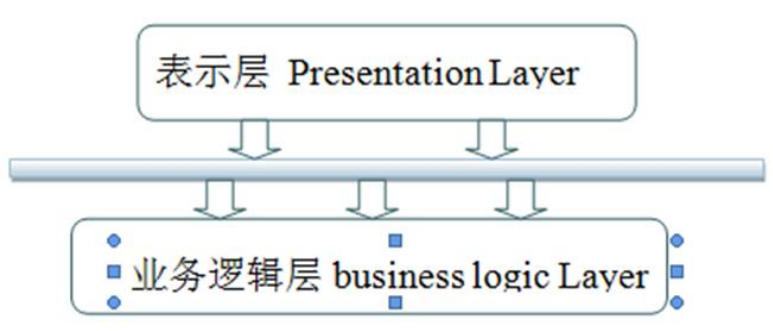

Memu
TaxCalculator
base
rebuildTaxRateTable()
setBase()
calculate()
taxRateTable
TaxRateTable
nRank
fullRate
calFullRate()
sortTaxRateTable()
test()
insertRank()
deleteRank()
changeRank()
calculate()
List rate
Rate
start
rate
getStart()
getRate()
calculate()

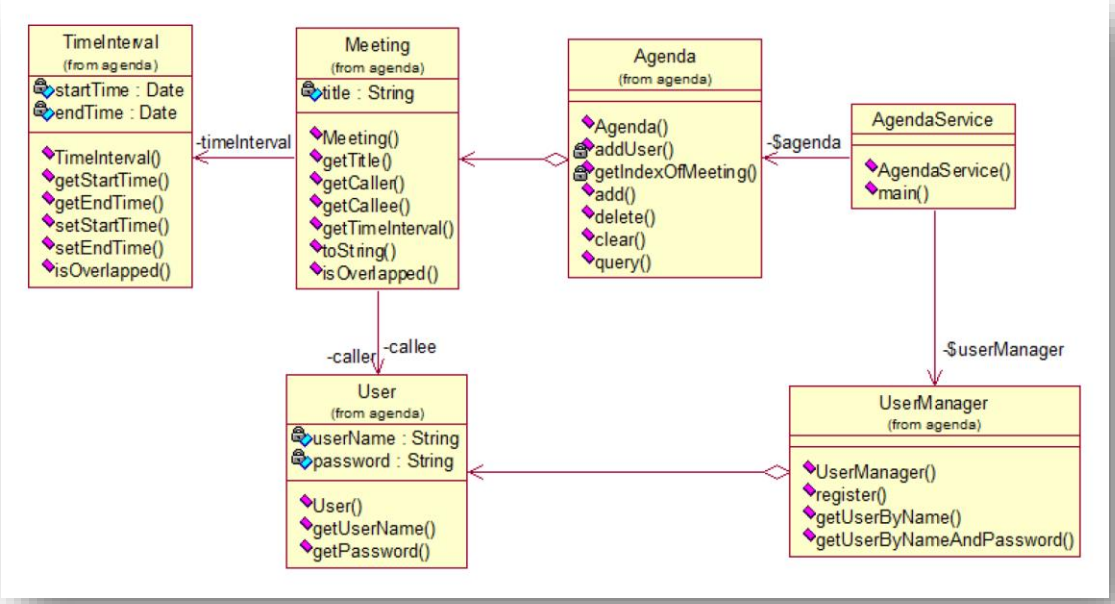
中山大學
SUN YAT-SEN UNIVERSITY

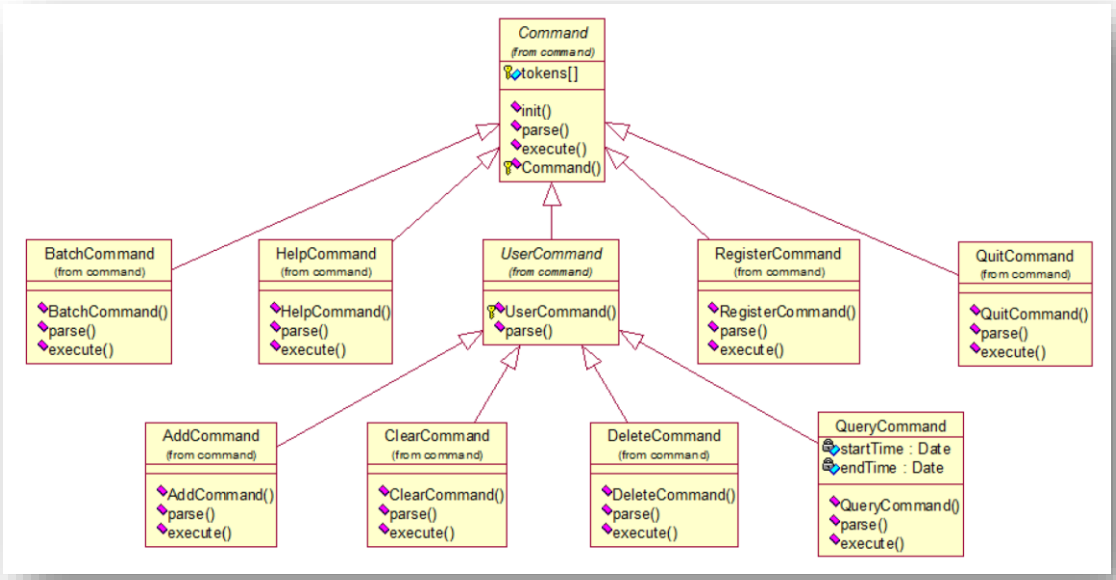
Version 3.00; Sep. 28, 2017

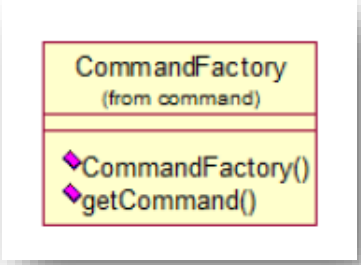
编译原理实验

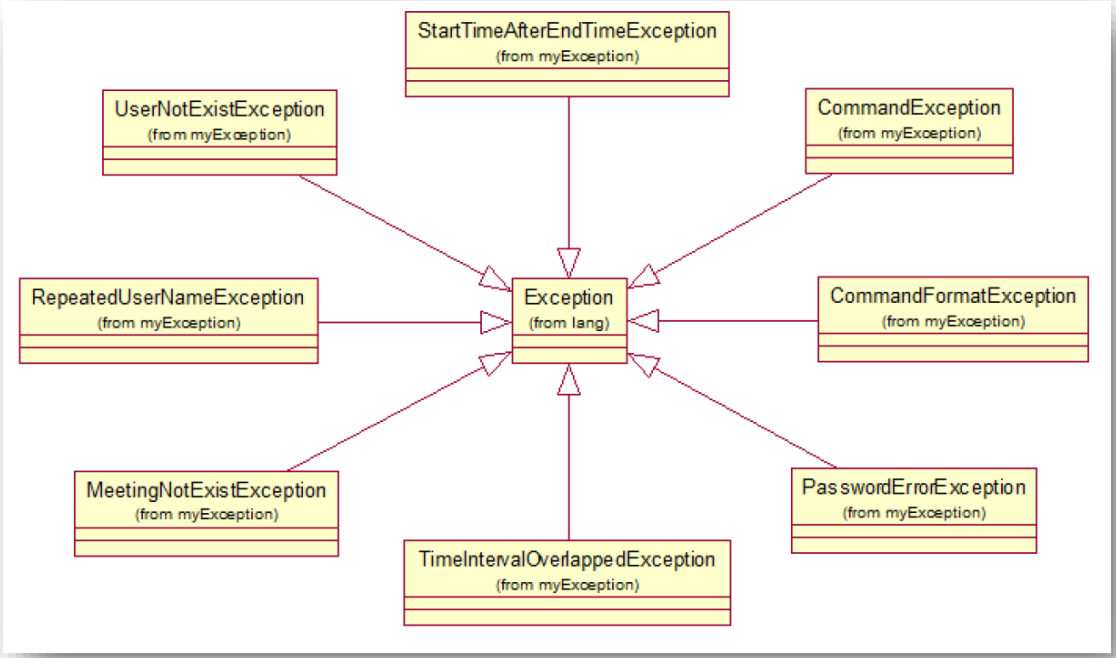
议程管理系统 *Agenda*






Command
(from command)
tokens[]
init()
parse()
execute()
Command()
BatchCommand
(from command)
BatchCommand()
parse()
execute()
HelpCommand
(from command)
HelpCommand()
parse()
execute()
UserCommand
(from command)
UserCommand()
parse()
RegisterCommand
(from command)
RegisterCommand()
parse()
execute()
QuitCommand
(from command)
QuitCommand()
parse()
execute()
AddCommand
(from command)
AddCommand()
parse()
execute()
ClearCommand
(from command)
ClearCommand()
parse()
execute()
DeleteCommand
(from command)
DeleteCommand()
parse()
execute()
QueryCommand
startTime : Date
endTime : Date
QueryCommand()
parse()
execute()


CommandFactory
(from command)
CommandFactory()
getCommand()


StartTimeAfterEndTimeException
(from myException)
UserNotExistException
(from myException)
CommandException
(from myException)
RepeatedUserNameException
(from myException)
Exception
(from lang)
CommandFormatException
(from myException)
MeetingNotExistException
(from myException)
PasswordErrorException
(from myException)
TimeIntervalOverlappedException
(from myException)