Lab4 文法设计实验

姓名/学号：宋尚儒/1120180717

实验目的

本次实验的主要目的是了解程序设计语言的演化过程和相关标准的制定过程， 深入理解与编译实现有关的形式语言理论，熟练掌握文法及其相关的概念，并能够使用文法对给定的语言进行描述，为后面的词法分析和语法分析做准备。

实验内容

（1）阅读附件提供的 C 语言和 Java 语言的规范草稿，了解语言规范化定义应包括的具体内容。
（2）选定 C 语言子集，并使用 BNF 表示方法文法进行描述， 要求至少包括表达式、 赋值语句、分支语句和循环语句； 或者设计一个新的程序设计语言，并使用文法对该语言的词法规则和文法规则进行描述。
　以上语言定义首先要给出所使用的字母表，在此基础上使用 2 型文法描述语法规则。

实验过程

字母表

终结符	含义
IntegerConstant	整型常量
FloatingConstant	浮点型常量
Identifier	标识符
CharacterConstant	字符型常量
StringLiteral	字符串型常量

此外终结符还包括

- auto、break、case……等关键词
- [、]、+、-……等算符

非终结符	含义
trans_unit	程序
decl_ex	声明或函数定义
func_def	函数定义
decl	声明
init_declarator_list	声明变量初始化列表，用“,”分隔
init_declarator	变量初始化赋值
declarator	声明符，存在指针形式
pointer	指针前缀
direct_declarator	直接声明符
decl_spec	声明类型
type_spec	具体的类型关键词
parameter_list	参数声明列表
parameter_decl	参数声明
identifier_list	标识符列表
init	初始化值语句
init_list	初始化语句列表
exp	表达式，优先级最低，表示包括','的总表达式
assign_exp	赋值表达式，优先级高于','，低于其他
assign_operator	赋值算符
con_exp	条件表达式
unary_exp	一元前缀表达式
unary_operator	一元算符
postfix_exp	带有后缀的表达式
augument_exp_list	变量列表
pri_exp	表达式，有最高优先级，表示各类常量或()
multiplicative_exp	表达式，优先级*、/、%
additive_exp	表达式，优先级+、-
shift_exp	表达式，优先级>>、<<
relational_exp	表达式，优先级>、<、>=、<=
equality_exp	表达式，优先级==、!=

非终结符	含义
and_exp	表达式，优先级&
exclusive_or_exp	表达式，优先级^
inclusive_or_exp	表达式，优先级|
logical_and_exp	表达式，优先级&&
logical_or_exp	表达式，优先级||
block_item_list	复合语句成员列表
block_item	复合语句成员
state	语句
exp_state	表达式语句
select_state	选择语句
iteration_state	循环语句
jump_state	跳转语句

文法设计

该部分由四部分组成，分步讲解各部分文法，为方便查阅，会先以分行的形式表达或关系，终结符会加粗表示，在每部分的最后给出BNF表示，为防止混淆，会用`<>`表示非终结符，`''`表示终结符，符号中间用空格分开表示连接，用`|`表示或关系

全局结构

程序仅由数量大于等于一的声明或函数定义构成，函数定义必须包括类型声明、声明符、复合语句（包括{}的函数体）

- trans_unit
  - decl_ex
  - decl_ex trans_unit
- decl_ex
  - func_def
  - decl
- func_def
  - decl_spec declarator comp_state

设计BNF语句为

```
<trans_unit> -> <decl_ex> | <decl_ex trans_unit>

<decl_ex> -> <func_def> | <decl>

<func_def> -> <decl_spec> <declarator> <comp_state>
```

声明

在本设计中，由于不考虑实现结构体、枚举等复杂的结构，并且不准备对存储特性做进一步区分，仅对一般数据类型、数组和函数声明进行文法设计，支持在一个声明语句内声明多个变量并初始化。

- decl
  - decl_spec init_declarator_list **;**
  - decl_spec **;**
- init_declarator_list
  - init_declarator
  - init_declarator **,** init_declarator_list
- init_declarator
  - declarator
  - declarator **=** init
- declarator
  - direct_declarator
  - point direct_declarator
- pointer
  - ***** .
  - ***** pointer
- direct_declarator
  - **Identifier**
  - **(** declarator **)**
  - direct_declarator **[ ]**
  - direct_declarator **[** assign_exp **]**
  - direct_declarator **( )**
  - direct_declarator **(** parameter_list **)**
  - direct_declarator **(** identifier_list **)**
- decl_spec
  - type_spec
  - type_spec decl_spec
- type_spec
  - **void** | **char** | **short** | **int** | **long** | **float** | **double** | **signed** | **unsigned**
- parameter_list
  - parameter_decl
  - parameter_decl **,** parameter_list
- parameter_decl
  - decl_spec declarator
- identifier_list
  - **Identifier**
  - **Identifier ,** identifier_list
- init
  - assign_exp
  - **{** init_list **}**
- init_list
  - init
  - init **,** init_list

设计BNF语句为

```
<decl> -> <decl_spec> <init_declarator_list> ';' | <decl_spec> ';'

<init_declarator_list> -> <init_declarator> | <init_declarator> ','
<init_declarator_list>

<init_declarator> -> <declarator> | <declarator> '=' <init>

<declarator> -> <direct_declarator> | <point> <direct_declarator>

<pointer> -> '*' | '*' <pointer>

<direct_declarator> -> 'Identifier' | <direct_declarator> '[' ']' |
<direct_declarator> '[' <assign_exp> ']' | <direct_declarator> '(' ')' |
<direct_declarator> '(' <parameter_list> ')' | <direct_declarator> '('
<identifier_list> ')'

<decl_spec> -> <type_spec> | <type_spec> <decl_spec>

<type_spec> -> 'void' | 'char' | 'short' | 'int' | 'long' | 'float' | 'double' |
'signed' | 'unsigned'

<parameter_list> -> <parameter_decl> | <parameter_decl> <parameter_list>

<parameter_list> -> <decl_spec> <declarator>

<identifier_list> -> 'Identifier' | 'Identifier' <identifier_list>

<init> - > <assign_exp> | '{' <init_list> '}'

<init_list> -> <init> |
```

表达式

为在一定程度上防止二义性，表达式的算符存在优先级，故设置不同的非终结符以实现对不同优先级算符的识别，各非终结符对应的含义已经在之前说明，优先级自高到低分别为：

- pri_exp
- postfix_exp
- unary_exp
- multiplicative_exp
- additive_exp
- shift_exp
- relational_exp
- equality_exp
- and_exp
- exclusive_or_exp
- inclusive_or_exp
- logical_and_exp
- logical_or_exp
- con_exp
- assign_exp
- exp

可简化表示各BNF范式如下

- exp
  - assign_exp
  - assign_exp **,** exp
- assign_exp
  - con_exp
  - unary_exp assign_operator assign_exp
- assign_operator
  - **=** | ***=** | **/=** | **%=** | **+=** | **-=** | **<<=** | **>>=** | **&=** | **^=** | **|=**
- con_exp
  - logical_or_exp
  - logical_or_exp **?** expression **:** con_exp
- unary_exp
  - postfix_exp
  - **++** unary_exp
  - **--** unary_exp
  - unary_operator unary_exp
  - **sizeof** unary_exp
- unary_operator
  - **&** | ***** | **+** | **-** | **~** | **!**
- postfix_exp
  - pri_exp
  - postfix_exp **[** exp **]**
  - postfix_exp **(** argument_exp_list **)**
  - postfix_exp **(** **)**
  - postfix_exp **. Identifier**
  - postfix_exp **-> Identifier**
  - postfix_exp **++**
  - postfix_exp **--**
- argument_exp_list
  - assign_exp
  - assign_exp **,** argument_exp_list
- pri_exp
- **Identifier**
  - **IntegerConstant**
  - **FloatingConstant**
  - **CharacterConstant**
  - **StringLiteral**
  - **(** exp **)**
- logical_or_exp
  - logical_and_exp
  - logical_and_exp **||** logical_or_exp
- logical_and_exp
  - inclusive_or_exp
  - inclusive_or_exp **&&** logical_and_exp
- inclusive_or_exp
  - exclusive_or_exp
  - exclusive_or_exp **|** inclusive_and_exp
- exclusive_or_exp

    - and_exp
    - and_exp ^ exclusive_or_exp
- and_exp
    - equality_exp
    - equality_exp **&** and_exp
- equality_exp
    - relational_exp
    - relational_exp **==** equality_exp
    - relational_exp **!=** equality_exp
- relational_exp
    - shift_exp
    - shift_exp **<** relational_exp
    - shift_exp **>** relational_exp
    - shift_exp **<=** relational_exp
    - shift_exp **>=** relational_exp
- shift_exp
    - additive_exp
    - additive_exp **<<** shift_exp
    - additive_exp **>>** shift_exp
- additive_exp
    - multiplicative_exp
    - multiplicative_exp **+** additive_exp
    - multiplicative_exp **-** additive_exp
- multiplicative_exp
- unary_exp
    - unary_exp ***** unary_exp
    - unary_exp **/** unary_exp
    - unary_exp **%** unary_exp

设计BNF语句为

```
<exp> -> <assign_exp>  |  <assign_exp> <exp>

<assign_exp> -> <con_exp>  |  <unary_exp> <assign_operator> <assign_exp>

<assign_operator> -> '=' | '*=' | '/=' | '%=' | '+=' | '-=' | '<<=' | '>>=' |
'&=' | '^=' | '|='

<con_exp> -> <logical_or_exp>  |  <logical_or_exp> '?'  <exp> ':' <con_exp>

<unary_exp> -> <postfix_exp>  | '++' <unary_exp> | '--' <unary_exp> |
<unary_operator> <unary_exp> | 'sizeof' <unary_exp>

<unary_operator> -> '&' | '*' | '+' | '-' | '~' | '!'

<postfix_exp> -> <pri_exp>  | <postfix_exp> '[' <exp> ']'  | <postfix_exp> '('
<argument_exp_list> ')' | <postfix_exp> '[' ']' | <postfix_exp> '.'
'Indentifier' | <postfix_exp> '->' 'Identifier' | <postfix_exp> '++'|
<postfix_exp> '--'

<augument_exp_list> -> <assign_exp> <augument_exp_list>
```

```
<pri_exp> -> 'Identifier' | 'IntegerConstant' | 'FloatingConstant' |
'CharacterConstant' | 'StringLiteral' | '(' <exp> ')'

<logical_or_exp> -> <logical_and_exp> | <logical_and_exp> '||' <logical_or_exp>

<logical_and_exp> -> <inclusive_or_exp> | <inclusive_or_exp> '&&'
<logical_and_exp>

<inclusive_or_exp> -> <exclusive_or_exp> | <exclusive_or_exp> '|'
<inclusive_or_exp>

<exclusive_or_exp> -> <and_exp> | <and_exp> '^' <exclusive_or_exp>

<and_exp> -> <equality_exp> | <equality_exp> '&' <and_exp>

<equality_exp> -> <relational_exp> | <relational_exp> '==' <equality_exp> |
<relational_exp> '!=' <equality_exp>

<relational_exp> -> <shift_exp> | <shift_exp> '<' <relational_exp> | <shift_exp>
'>' <relational_exp> | <shift_exp> '<=' <relational_exp> | <shift_exp> '>='
<relational_exp>

<additive_exp> -> <multiplicative_exp> | <multiplicative_exp> '+' <additive_exp>
| <multiplicative_exp> '-' <additive_exp>

<multiplicative_exp> -> <unary_exp> | <unary_exp> '*' <multiplicative_exp> |
<unary_exp> '/' <multiplicative_exp> | <unary_exp> '%' <multiplicative_exp>
```

语句

每个函数定义必须包括一个复合语句，复合语句中存在若干声明或语句，语句可以是复合语句、选择语句、循环语句等等。

在本设计中

- 循环语句考虑while、dowhile、for循环，for循环的第一个语句支持声明语句
- 选择语句考虑if、else形式，不考虑switch选择
- 考虑跳转语句

可简化表示各BNF范式如下

- comp_state
  - **{ }**
  - **{** block_item_list **}**
- block_item_list
  - block_item
  - block_item block_item_list
- block_item
  - declaration
  - state
- state
  - comp_state
  - labeled_state
  - exp_state

    - select_state
    - iteration_state
    - jump_state
- labeled_state
    - **Identifier :** state
    - **case** con_exp **:** state
    - **Identifier :** state
- exp_state
    - **;**
    - exp **;**
- select_state
    - **if (** exp **)** state
    - **if (** exp **)** state **else** state
    - **switch (** exp **)** state
- iteration_state
    - **while (** exp **)** state
    - **do** state **while (** exp **) ;**
    - **for (** exp_state exp_state exp **)** state
    - **for (** exp_state exp_state **)** state
    - **for (** declaration exp_state exp **)** state
    - **for (** declaration exp_state **)** state
- jump_state
    - **goto Identifier ;**
    - **continue ;**
    - **break ;**
    - **return** exp **;**
    - **return ;**

设计BNF语句为

```
<comp_state> -> '{' <block_item_list> '}'  |  '{' '}'

<block_item_list> -> <block_item> | <block_item block_item_list>

<block_item> -> <declaration> | <state>

<state> -> <comp_state> | <labeled_state> | <exp_state> | <select_state> |
<iteration_state> | <jump_state>

<labeled_state> -> 'Identifier' ':' <state> | 'case' <con_exp> ':' <state> |
'Identifier' ':' <state>

<exp_state> -> ';' | <exp> ';'

<select_state> -> 'if' '(' <exp> ')' <state> | 'if' '(' <exp> ')' <state> 'else'
<state>

<iteration_state> -> 'while' '(' <exp> ')' <state> | 'do' <state> 'while' '('
<exp> ')'  | 'for' '(' <exp_state> <exp_state> <exp> ')' <state>   | 'for' '('
<exp_state> <exp_state> ')' <state>   | 'for' '(' <declaration> <exp_state>
<exp> ')' <state>   | 'for' '(' <declaration> <exp_state> ')' <state>
```

```
<jump_state> -> 'goto' 'Identifier' | 'continue' | 'break' | 'return' | 'return'
<exp>
```

实验心得体会