实验四、文法设计实验

> *学号：1120180207*
>
> *姓名：唐小娟*
>
> *班级：07111801*

1. 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。

2. 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。

本实验选定C语言子集，采用BNF表示方法，使用2型文法描述语法规则。

3. 语言规范

3.1 定义

语言标准规范是每个语言的说明文档，定义了语言的字符集、词法规则、语法规则和语义规则，也包括了对程序结构、编译过程、标准库程序以及语言实现等各方面的内容。

3.2 个人理解

规范，顾名思义，是规则的典范，也就是标准。每个人对自己编写代码都有一定的偏好，如果要开发一种大家通用的语言，首先需要约定标准，比如变量、表达式的使用等等，这样在遇到分歧的时候，才会达成共识。制定规范，也是给使用者在遇到不明确的问题时参考的资料，方便使用，也会给出限制，这样才能在整个使用语言的环境中有良好的秩序，类似我们社会中的法律与规则。

4. 文法设计

该文法设计参考的是C11规范，为了简便，实验关注比较常用的语法规则。

4.1 表达式

4.1.1 Primary expression

基本表达式包括了标识符、常量、字符串和加了括号的表达式。

```
<primary-expression> ::= <identifier>
                    |constant
                    |string-constant
                    |"(" <expression>")"
```

4.1.2 Postfix expression

后缀表达式包括基本表达式、数组形式、函数调用（一系列参数复制表达式）、结构体成员变量访问，自增自减。<assignment-expression>的文法规则在文后。

```
<postfix-expression> := <primary-expression>
                     |<postfix-expression> "[" <expression> "]"
                     |<postfix-expression> "(" <assignment-
expression-list> ")"
                     |<postfix-expression> "." <identifier>
                     |<postfix-expression> "->" <identifier>
                     |<postfix-expression> "++"
                     |<postfix-expression> "--"

<assignment-expression-list> := <assignment-expression>
                         |<assignment-expression-list> ","
<assignment-expression>
```

4.1.3 Unary expression

一元表达式包括后缀表达式、前缀自增自减、取地址、指向地址、一元运算、sizeof。注意sizeof是一元运算符，而不是一个函数。所以sizeof其后的参数也可以不加括号。

```
<unary-expression> := <postfix-expression>
                    |"++" <unary-expression>
                    |"--" <unary-expression>
                    |<unary-expression> <cast-expression>
                    |"sizeof" <unary-expression>
                    |"sizeof" "(" <type-name> ")"

<unary-operator> := "&"|"*"|"+"|"-"|"~"|"!"
```

4.1.4 Cast expression

强制转换表达式包括隐式和显式。

```
<cast-expression> := <unary-expression>
                   |"(" <type-name> ")" <cast-expression>
```

4.1.5 Multiplicative expression

乘法表达式包括强制转换表达式、乘法、除法、取余。

```
<multiplicative-expression> := <cast-expression>
                             |<multiplicative-expression> "*"
<cast-expression>
                             |<multiplicative-expression> "/"
<cast-expression>
                             |<multiplicative-expression> "%"
<cast-expression>
```

4.1.6 Additive expression

这里有个很巧妙的点，加法表达式可以包括乘法表达式，说明乘法的优先级要高于加法。除此之外，additive-expression是左递归式，满足了左结合律。

```
<additive-expression> := <multiplicative-expression>
                       |<additive-expression> "+"
<multiplicative-expression>
                       |<additive-expression> "-"
<multiplicative-expression>
```

4.1.7 Shift expression

同样，移位表达式具有左结合的特点，并且加法优先级要高于移位运算符。

```
<shift-expression> := additive-expression
                    |<shift-expression> "<<" <additive-
expression>
                    |<shift-expression> ">>" <additive-
expression>
```

4.1.8 Relational expression

关系表达式

```
<relational-expression> := shift-expression
                         |<relational-expression> "<" <shift-
expression>
                         |<relational-expression> ">" <shift-
expression>
                         |<relational-expression> "<=" <shift-
expression>
                         |<relational-expression> ">=" <shift-
expression>
```

4.1.9 Equality expression

相等表达式

```
<equality-expression> := <relational-expression>
                         |<equality-expression> "==" <relational-
expression>
                         |<equality-expression> "!=" <relational-
expression>
```

4.1.10 AND expression

与表达式

```
<AND-expression> := equality-expression
                 |<AND-expression> "&" <equality-expression>
```

4.1.11 Exclusive OR expression

异或表达式

```
<exclusive-OR-expression> := <AND-expression>
                           |<exclusive-OR-expression> ^ <AND-
expression>
```

4.1.12 Inclusive OR expression

或表达式

```
<inclusive-OR-expression> := <exclusive-OR-expression>
                           |<inclusive-OR-expression> "|"
<exclusive-OR-expression>
```

4.1.13 Logical AND expression

逻辑与表达式

```
<logical-AND-expression> := <inclusive-OR-expression>
                          |<logical-AND-expression> "&&"
<inclusive-OR-expression>
```

4.1.14 Logical OR expression

逻辑或表达式

```
<logical-OR-expression> := <logical-AND-expression>
                         |<logical-OR-expression> "||" <logical-
AND-expression>
```

4.1.15 Conditional expression

条件表达式注意是右结合。

```
<conditional-expression> := <logical-OR-expression>
                          |<logical-OR-expression> ? <expression>
: <conditional-expression>
```

4.1.16 Assignment expression

赋值表达式，同样也是右结合

```
<assignment-expression> := <conditional-expression>
                         |<unary-expression> <assignment-
operator> <assignment-expression>

<assignment-operator> := "="|"*="|"/="|"%="|"+="|"-="|"
<<="|">>="|"&="|"^="|"|="
```

4.1.17 Comma expression

逗号表达式是左结合的。

```
<comma-expression> := <assignment-expression>
                    |<expression> "," <assignment-expression>
```

4.2 常数表达式

常数表达式可以在编译的时候就给出结果而不必在运行时候，而且也可以用在常量的地方。

```
<constant-expression> := <conditional-expression>
```

4.3 声明

```
<declaration> := <declaration-specifiers> ";"
                |<declaration-specifiers> <init-declarator-list> ";"

<declaration-specifiers> := <storage-class-specifier>
                          |<storage-class-specifier> <declaration-specifiers>
                          |<type-specifier>
                          |<type-specifier> <declaration-specifiers>
                          |<type-qualifier>
                          |<type-qualifier> <declaration-specifiers>
                          |<function-specifier>
                          |<function-specifier> <declaration-specifiers>


<init-declarator-list> := <init-declarator>
                          |<init-declarator-list> "," <init-declarator>

<init-declarator> := <declarator>
                     |<declarator> "=" <initializer>
```

4.3.1 Storage class specifier

存储类关键词

```
<storage-class-specifier> := "typedef"
                             |"extern"
                             |"static"
                             |"auto"
                             |"register"
```

4.3.2 Type specifiers

类型说明

```
<type-specifier> := "void"
                  |"char"
```

```
                |"short"
                |"int"
                |"long"
                |"float"
                |"double"
                |"signed"
                |"unsigned"
                |<struct-or-union-specifier>
                |<enum-specifier>
                |<typedef-name>

//结构体或者union体，可以是匿名对象、非匿名对象。
<struct-or-union-specifier> := <struct-or-union> "{" <struct-
declaration-list> "}"
                        |<struct-or-union> <identifier> "{"
<struct-declaration-list> "}"
                        |<struct-or-union> <identifier>

<struct-or-union> := "struct"|"union"

//结构体成员变量
<struct-declaration-list> := <struct-declaration>
                            |<struct-declaration-list> <struct-
declaration>

<struct-declaration> := <specifier-qualifier-list> ";"
                    |<specifier-qualifier-list> <struct-
declarator-list> ";"

<specifier-qualifier-list> := <type-specifier>
                            |<type-specifier> <specifier-
qualifier-list>
                            |<type-qualifier>
                            |<type-qualifier> <specifier-
qualifier-list>

<struct-declarator-list> := <struct-declarator>
                            |<struct-declarator-list> ","
<struct-declarator>

<struct-declarator> :=  <declarator>
                    //位域
                    |<declarator> ":" <constant-expression>
                    |":" <constant-expression>

//enum体
<enum-specifier> := "enum" <identifier> "{" <enumerator-list>
"}"
                |"enum" "{" <enumerator-list> "}"
                |"enum" <identifier> "{" <enumerator-list> ","
"}"
                |"enum" "{" <enumerator-list> "," "}"
```

```
                |"enum" <identifier>

//enum成员变量
<enumerator-list> := <enumerator>
                    |<enumerator-list> "," <enumerator>

<enumerator> := <enumeration-constant>
            |<enumeration-constant> "=" <constant-expression>

<enumeration-constant> := <identifier>
```

4.3.3 Type qualifier

类型限定符

```
<type-qualifier> := "const"|"restrict"|"volatile"
```

4.3.4 Function specifier

函数说明符

```
<function-specifier> := "inline"
```

4.3.5 Declarators

说明符，为了简便，这里采用EBNF。

```
<declarator> := [<pointer>] <direct-declarator>

<direct-declarator> := identifier
            |"(" <declarator> ")"
            //数组声明符
            |<direct-declarator> "[" [<type-qualifier-list>]
[<assignment-expression>] "]"
            //函数声明符
            |<direct-declarator> "(" <parameter-type-list> ")"
            |<direct-declarator> "(" [<identifier-list>] ")"

//指针形式
<pointer> := "*" [<type-qualifier-list>]
            |"*" [<type-qualifier-list>] <pointer>

//一系列类型限定符
<type-qualifier-list> := <type-qualifier>
                        |<type-qualifier-list> <type-qualifier>

<parameter-type-list> := <parameter-list>

//参数列表
```

```
<parameter-list> := <parameter-declaration>
                |<parameter-list> "," <parameter-declaration>

<parameter-declaration> := <declaration-specifiers>
<declarator>
                        |<declaration-specifiers>

<identifier-list> := <identifier>
                    |<identifier-list> "," <identifier>
```

4.3.6 Type names

类型名，不仅仅有int、char这样的，还有指针int *、int *[]这样的。

```
<type-name> := <specifier-qualifier-list> [<abstract-
declarator>]

<abstract-declarator> := <pointer>
                        |<pointer> <direct-abstract-declarator>

<direct-abstract-declarator> := "(" <abstract-declarator> ")"
                            |[<direct-abstract-declarator>] "["
[<type-qualifier-list>] [<assignment-expression>] "]"
                            |[<direct-abstract-declarator>] "["
"static" [<type-qualifier-list>] [<assignment-expression>] "]"
                            |[<direct-abstract-declarator>] "["
[<type-qualifier-list>] "static" [<assignment-expression>] "]"
                            |[<direct-abstract-declarator>] "["
"*" "]"
                            |[<direct-abstract-declarator>] "("
[<parameter-type-list>] ")"

```

4.3.7 Initialization

初始化

```
<initializer> := <assignment-expression>
                |"{" <initializer-list> "}"
                |"{" <initializer-list> "," "}"

<initializer-list> := [<designation>] <initializer>
                    |<initializer-list> "," [<designation>]
<initializer>

<designation> := <designator-list> "="

<designator-list> := <designator>
                    |<designator-list> <designator>

```

```
<designator> := "[" <constant-expression> "]"
             |"." <identifier>
```

4.4 语句块

```
<statement> := <labeled-statement>
            |<compound-statement>
            |<expression-statement>
            |<selection-statement>
            |<iteration-statement>
            |<jump-statement>
```

4.4.1 Labeled statement

有标记的语句，出现在switch语句中

```
<labeled-statement> := <identifier> ":" <statement>
                    |"case" <constant-expression> ":"
<statement>
                    |"default" ":" <statement>
```

4.4.2 Compound statement

复合语句，也就是花括号括起来的。

```
<compound-statement> := "{" [<block-item-list>] "}"

<block-item-list> := <block-item>
                |<block-item-list> <block-item>

<block-item> := <declaration>
            |<statement>
```

4.4.3 Expression statement

表达式语句

```
<expression-statement> := [<expression>] ";"
```

4.4.4 Selection statement

选择语句，if和switch语句。

```
<selection-statement> := "if" "(" <expression> ")" <statement>
                        |"if" "(" <expression> ")" <statement>
"else" <statement>
                        |"switch" "(" <expression> ")"
<statement>
```

4.4.5 Iteration statements

循环语句，包括for和while循环

```
<iteration-statement> := "while" "(" <expression> ")"
<statement>
                       |"do" <statement> "while" "("
<expression> ")" ";"
                       |"for" "(" [<expression>] ";"
[<expression>] ";" [<expression>] ")" <statement>
                       |"for" "(" <declaration> [<expression>]
";" [<expression>] ")" <statement>
```

4.4.6 Jump statements

跳转语句

```
<jump-statement> := "goto" <identifier> ";"
                  |"continue" ";"
                  |"break" ";"
                  |"return" [<expression>] ";"
```

4.4.7 Function definitions

函数定义

```
<function-definition> := <declaration-specifiers> <declarator>
[<declaration-list>] <compound-statement>

<declaration-list> := <declaration>
                    |<declaration-list> <declaration>
```

5. 心得体会

在本实验中，我收获到了很多：

1. 通过借助C11的规范理解了C语言以前没意识的语法，比如位域，有些信息存储时不需要占用一个完整的字节，而只需要占用几个二进制位。这让我对C这门编程语言有了很深的认识。
2. 在写BNF的过程中，我通过例子并且结合自己的思考，理解了在B规则的书写中如何体现优先级以及结合律的特点。
3. 单看C语言的规范，就知道要制定好一门语言的规范很不容易，很多细节以及很多方面都要考虑到。而语言的规范在语言的使用以及编译器的设计过程中又是十分关键的作用。
4. 文法设计的过程，让我对编译器有了进一步的认识，为后续语法分析奠定了基础。