北京航空航天大學

BEIHANG UNIVERSITY

扩充 C0 文法解释

院（系）名称	计算机学院
专业名称	计算机科学与技术
学号	16061170
姓名	宋卓洋

2018 年 11 月

扩充 C0 文法解释

1. 符号集与保留字

1.1 符号集

{ ‘+’, ‘-’, ‘*’, ‘/’, ‘<’, ‘<=’, ‘>’, ‘>=’, ‘==’, ‘!=’, ‘=’, ‘(’, ‘)’, ‘[‘, ‘]’, ‘{‘, ‘}’, ‘,’, ‘;’, ‘:’, ‘”, ‘””, ‘a’, ... , ‘z’, ‘A’, ... , ‘Z’, ‘0’, ... , ‘9’, ‘_’}

1.2 保留字

{ const, void, int, char, if, while, switch, case, default, return, main, printf, scanf}

2. 文法解释

- *＜加法运算符＞ ::= + | -*

解释：加法运算符可以为’+’或’-’。

- *＜乘法运算符＞ ::= * | /*

解释：乘法运算符可以为’*’或’/’。

- *＜关系运算符＞ ::= < | <= | > | >= | != | ==*

解释：关系运算符可以为’<’、’<=’、’>’、’>=’、’!=’、’==’。

- *＜字母＞ ::=_ | a | . . . | z | A | . . . | Z*

解释：字母可以为大小写字母以及下划线’_’。

- *＜数字＞ ::= 0 | ＜非零数字＞*

解释：数字可以为 0 或非零数字。

- *＜非零数字＞ ::= 1 | . . . | 9*

解释：非零数字可以为 1 至 9 中任意一个。

- *＜字符＞ ::= '＜加法运算符＞' | '＜乘法运算符＞' | '＜字母＞' | '＜数字＞'*

解释：规定了字符的格式。字符由单引号标识边界，内部可以为单个的加法运算符、乘法运算符、字母或数字。要注意字符的不包含其他的 ASCII 字符，与字符串的范围进行区别。

例子：‘+’ ‘-’ ‘7’ ‘a’

- *＜字符串＞ ::= "｛十进制编码为 32,33,35-126 的 ASCII 字符｝"*

解释：字符串由双引号标识边界，内部为任意数量的十进制编码为 **32,33,35-126 的 ASCII 字符**。注意不存在转义字符的概念。如“\n”按照原样输出，而不将其

转义为换行符。

例子：“fdafdafdafd dqafgv”“___21fewF[[].KJJHFASF?&&(%$*dyj{}”

• *＜程序＞ ::= ［＜常量说明＞］［＜变量说明＞］{＜有返回值函数定义＞|＜无返回值函数定义＞}＜主函数＞*

解释：规定了程序的基本结构。程序由常量说明、变量说明、函数定义、主函数依次按照顺序组成，规定了程序的整体结构。其中除主函数为必选项外其他都为可选部分。常量说明和变量说明都分别最多存在一个。函数定义分为有返回值函数定义和无返回值定义，在文法上两者间无顺序要求，且都可存在多个。注意这几个部分之间的顺序关系。

• *＜常量说明＞::= const＜常量定义＞;{ const＜常量定义＞;}*

解释：规定了常量说明的整体结构。常量说明由一至多个“常量说明部分”构成。每个“常量说明部分”都要由保留字“const ”作为前缀，后接一个常量定义，最后以分号作为该部分的结尾。

• *＜常量定义＞ ::= int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}*

| char ＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}

解释：规定了定义每个常量时的结构。常量定义分为整型常量定义和字符型常量定义。二者分别由保留字“int”和“char”作为前缀。整型常量定义可以对一至多个整型常量进行定义。定义的格式为：*＜标识符＞＝＜整数＞*。每个定义间以逗号作为分隔。字符型常量定义可以对一至多个字符型常量进行定义。定义的格式为：*＜标识符＞＝＜字符＞*。每个定义间以逗号作为分隔。注意每次仅能对同一种常量进行定义，并且需要对常量进行赋值操作。

例子：const char c = ‘a’; const int i = 1, j = 2, k = 3;

• *＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝| 0*

• *＜整数＞ ::= ［＋｜－］＜无符号整数＞*

解释：

无符号整数可为 0 或以非零数字作为前缀后接多个数字。含有前缀零的数字为非法。

整数由无符号前接至多一个加号或减号构成。

例子：

• 整数：+10101, -10801481, 124566

• 无符号整数：21676, 4566,77

• ***＜标识符＞ ::= ＜字母＞{＜字母＞|＜数字＞}***

解释：标识符以字母开头，后面由任意个字母或数字构成。

• ***＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞***

解释：声明头部由“int”或“char”作为开头，以单个标识符作为结尾。

• ***＜变量说明＞ ::=＜变量定义＞;{＜变量定义＞;}***

解释：变量说明由至少一个变量定义构成，每个变量定义间由分号分隔。

•＜变量定义＞ ::= ＜类型标识符＞(＜标识符＞|＜标识符＞'['＜无符号整数＞']'){,(＜标识符＞|＜标识符＞'['＜无符号整数＞']')}

解释：变量定义以类型标识符作为开头。后面可以定义至少一个变量或数组。若定义变量则由标识符构成；若定义数组则其文法格式为：*＜标识符＞'['＜无符号整数＞']'*。注意需要判断数组索引部分为大于零的无符号整数，不能为表达式，也不能为带“+”的整数。

例子：int i, k, HHH_[10]; char c, string[50];

• ***＜常量＞ ::= ＜整数＞|＜字符＞***

解释：常量为整数或字符。

• ***＜类型标识符＞ ::= int | char***

解释：类型标识符为“int”或“char”。

• ***＜有返回值函数定义＞ ::= ＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'***

解释：规定了有返回值函数定义的结构——有返回值函数定义由声明头部、参数表、复合语句三部分组成。声明头部作为该函数定义的开头，其后为由小括号括住的参数表，最后为花括号包裹的复合语句。

例子：int add(＜参数表＞){＜复合语句＞};char putc(＜参数表＞){＜复合语句＞};

• ***＜无返回值函数定义＞ ::= void＜标识符＞'('＜参数表＞')''{'＜复合语句＞'}'***

解释：规定了无返回值函数定义的结构——无返回值函数定义以“void”作为开

头，后面依次由函数名、参数表、复合语句三部分组成。函数名由标识符构成，参数表由小括号包围，复合语句由花括号包裹。

例子：void add(＜参数表＞){＜复合语句＞};

void putc(＜参数表＞){＜复合语句＞};

• ***＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞***

解释：复合语句由常量说明、变量说明、语句列依次构成。除语句列为必选项外，常量说明和变量说明为可选项，至多有一个。注意各部分的顺序。

• ***＜参数表＞ ::= ＜参数＞{,＜参数＞}| ＜空＞***

解释：参数表可以为空或者由至少一个构成，每个参数间由逗号分隔。

例子：int a, char b, int a

• ***＜参数＞ ::= ＜类型标识符＞＜标识符＞***

解释：参数由类型标识符加标识符构成。

例子：int a

注意：由以上几条规则可以看出函数声明的参数表部分无法表示数组结构，即类似于“int searchIn(int list[])”的结构是非法的。函数只能接受变量、常量形式的输入。

• ***＜主函数＞ ::= void main'('')''{'＜复合语句＞'}'***

解释：主函数由“***void main'('')***”作为前部，后接由花括号包围的复合语句。

• ***＜表达式＞ ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}***

解释：表达式由至少一个项组成。第一个项的前面可以由一个“+”或“-”，且该符号进作用于第一个项。其他项之间由加法运算符分隔。

• ***＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}***

解释：项由至少一个因子构成，每个因子间由乘法运算符分隔。

• ***＜因子＞ ::= ＜标识符＞｜＜标识符＞'['＜表达式＞']'|'('＜表达式＞')'｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞***

解读：因子可以为六种情况：

1. 单个标识符；
2. 标识符后接由中括号包围的表达式（数组）；

3. 由括号包围的表达式（嵌套表达）；
4. 整数；
5. 字符；
6. 有返回值函数的调用语句；

例子（表达式）：

+ 30 - 50 +1 * add(a, b) / b * (1 + sub(c, d) * a)

注意：根据之前对整数的定义，结合以上文法能够推导出类似于以下的形式：

+ + 3 - +3 * - 5

在这种情况下，考虑将正负号设置为仅对后面最近的数字产生作用。

注意：字符型和整型进行混合运算时有隐含的类型转换。

•<语句> ::= <条件语句>丨<循环语句>| '{'<语句列>'}'| <有返回值函数调用语句>; |<无返回值函数调用语句>;丨<赋值语句>;丨<读语句>;丨•<写语句>;丨<空>;|<情况语句>丨<返回语句>;

解释：语句分为 10 种情况。分别为条件语句、循环语句、语句块（由花括号包围的语句列）、函数调用语句（有返回值函数调用和无返回值函数调用）、赋值语句、读语句、写语句、情况语句、返回语句和空语句。注意其中函数调用语句、赋值语句、读写语句、情况语句、返回语句和空语句后部需以分号结尾，因为这些为单个的语句。

• <赋值语句> ::= <标识符>=<表达式>|<标识符>'['<表达式>']'=<表达式>

解释：赋值语句可将表达式的值赋给标识符和数组元素。表达式在右部，标识符和数组元素在左部，之间用等号连接。数组元素的结构为标识符后接用中括号包围的表达式。此外，该语句后部无分号。

例子：i = a + b 或 c[7] = i - c[i]

• <条件语句> ::= if '('<条件>')'<语句>

解释：条件语句由判断部分和行为部分组成。判断部分由“if”作为前部，后接由括号包围的条件。行为部分为语句。注意条件语句只有“如果”部分，无“否则”部分。

例子：

```
if(i == 0){
    i = 0;
}
```

- ***＜条件＞ ::= ＜表达式＞＜关系运算符＞＜表达式＞ | ＜表达式＞***

解释：条件语句由一至两个条件构成。若为两个表达式，则两者间由关系运算符分隔。

例子：1 >= 9 或 a+b <= add(c, d) 或 isDigit(a)

- ***＜循环语句＞ ::= while '('＜条件＞')'＜语句＞***

解释：循环语句由判断部分和循环体组成。判断部分由“while”作为前部，后接由括号包围的条件。循环体为语句。

例子：

```
while(i < 10){
    i = i + i;
}
```

- ***＜情况语句＞ ::= switch '('＜表达式＞')' '{'＜情况表＞＜缺省＞ '}'***

解释：情况语句由关键字声明和分类讨论两部分组成。情况语句由“switch”作为前部，后接由括号包围的表达式。分类讨论部分为情况表和缺省被花括号包围构成。

- ***＜情况表＞ ::= ＜情况子语句＞{＜情况子语句＞}***
- ***＜情况子语句＞ ::= case＜常量＞：＜语句＞***
- ***＜缺省＞ ::= default：＜语句＞|＜空＞***

解释：情况表由至少一个情况子语句组成。情况子语句由判断标签和行为部分组成。判断标签的结构为：“case”作为前部，分号作为后部，中间为常量。行为部分为语句。缺省可以为空，也可以为“default:”后接语句。与 C 语言不同的是在执行每个情况的行为部分语句后无需“break;”语句来忽略后面的情况，该文法默认只允许选择包含缺省在内的所有情况中的一个。

例子：

```
switch(i){
case 1: {
```

```
    i = i -1;
    }
case 2: i =0;
default:
}
```

- *＜有返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'*
- *＜无返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'*
- *＜值参数表＞ ::= ＜表达式＞{,＜表达式＞} | ＜空＞*

解释：函数调用语句结构为标识符后接由括号包围的值参数表。值参数表可以为空也可以为至少一个表达式，每个表达式以逗号作为分隔。

例子：add(a, b)

- *＜语句列＞ ::= {＜语句＞}*

解释：语句列为任意个语句，可以为空。

例子：

```
i = i + 1; j = i; if( i > 1) k = i + j;
```

- *＜读语句＞ ::= scanf '('＜标识符＞{,＜标识符＞}')'*

解释：读语句为对函数“scanf”的调用语句。其中值参数表不能为空，而且表达式只能为标识符（用于存储读取到的数据）。

例子：scanf(a,b,c)

- *＜写语句＞ ::= printf '(' ＜字符串＞,＜表达式＞ ')'| printf '('＜字符串＞ ')'| printf '('＜表达式＞')'*

解释：写语句为对函数“printf”的调用语句，写 1 至 2 个参数。可以单独写一个字符串或表达式；也可以同时依次写一个字符串和一个表达式（两者间以逗号分隔且顺序固定）。但是无法同时输出多个字符串或多个表达式。

例子：printf(string) 或 printf(a + b) 或 printf(string, a + v)

- *＜返回语句＞ ::= return['('＜表达式＞')']*

解释：返回语句可以仅为“return”，或者在其之后接由括号包围的表达式。

例子：return 或 return (a - b)

注意：每个语句文法规则中都未考虑后缀的分号，对于分号的使用在语句的文法

规则中统一规定。