**文法规则解读**

**1.＜加法运算符＞ ::= +｜-**

分析：定义了加减法运算符，优先级中

样例：A+B或A-B

**2.＜乘法运算符＞  ::= \*｜/**

分析：定义了乘法和除法运算，优先级高

样例：A\*B 或 A/B

**3.＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==**

分析：定义了六种关系运算，优先级低

样例：A<B或A<=B或A>B或A>=B或A==B或A!=B

**4.＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z**

分析：字母用于标识符的组成，所以包含\_下划线，需注意，该文法不区分标识符的大小写

样例：\_或a或A

**5.＜数字＞   ::= ０｜１｜．．．｜９**

分析：数字用于组成标识符和数字，其中标识符不能在名首

样例：0或1

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

分析：定义了字符的形式，是由’’单引号包含的加减乘除符号、字母和数字

样例：’+’或’\*’或’\_’或’A’或’1’

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

分析：字符串并非由字符组成的串，而是由””双引号包含的ASCII码32-126的字符串，除34双引号外，且不转义

样例：”abc”或”123”或””或”\n”

**8.＜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＞**

分析：这条文法定义了程序的结构，先是全局常量说明，然后是全局变量说明，之后是函数定义，最后是主函数

样例：

Const int A = 1;

Int B;

Int add(int x, int y):{

Return x + y;

}

void main(){

…

}

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

分析：该文法定义了常量说明规则，常量说明需统一在一块区域中

样例：

Const int A = 1, B = 2;

Const char C = ‘c’, D = ‘d’;

**10.＜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＞}**

分析：该文法定义了单句常量定义规则

样例：Const int A = 1, B = 2;或Const char C = ‘c’, D = ‘d’;

**11.＜无符号整数＞ ::= ＜数字＞｛＜数字＞｝**

分析：该文法定义了无符号整数，可以有多个前0

样例：000001或123456

**12.＜整数＞ ::= ［＋｜－］＜无符号整数＞**

分析：该文法定义了有符号整数，符号可有可无

样例：+00或00或-00

**13.＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝**

分析：该文法定义了标识符的构成，首字符不能是数字

样例：abc或abc123或\_123

**14.＜声明头部＞ ::= int＜标识符＞|char＜标识符＞**

分析：该文法定义了标识符的声明，有int和char两种类型

样例：int A或char A

**15.＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}**

分析：变量说明是一系列的变量定义，变量定义需在一块区域内完成

样例：

int A;

char B;

int C[10];

**16.＜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＞‘]’) } //＜无符号整数＞表示数组元素的个数，其值需大于0

分析：该文法定义了变量的定义，包括int、char和数组、字符串，其中数组和字符串的定义长度需大于0，并且变量定义不能赋值

样例：int A或int B[1]或int A,B,C[5]

**17.＜类型标识符＞ ::= int | char**

分析：定义了类型标识符的文法，只有数据类型，int和char

样例：int或char

**18.＜有返回值函数定义＞ ::= ＜声明头部＞‘(’＜参数表＞‘)’ ‘{’＜复合语句＞‘}’|＜声明头部＞‘{’＜复合语句＞‘}’**  //第一种选择为有参数的情况，第二种选择为无参数的情况

分析：该文法描述了有返回值的函数的格式，包含声明头部，被（）小括号包含的参数，被{}大括号包含的复合语句；如果没有参数则省略（）小括号。

样例：

Int add(int x, int y){

Return x+y;

}

或

Int return5{

Return 5;

}

**19.＜无返回值函数定义＞ ::= void＜标识符＞(’＜参数表＞‘)’‘{’＜复合语句＞‘}’| void＜标识符＞{’＜复合语句＞‘}’**//第一种选择为有参数的情况，第二种选择为无参数的情况

分析：该文法描述了无返回值的函数的格式，包含void和标识符，被（）小括号包含的参数，被{}大括号包含的复合语句；如果没有参数则省略（）小括号。

样例：

Void output(int A){

Printf(A);

}

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

分析：复合语句就是函数大括号内的内容，需要按照常量说明、变量说明、语句的顺序进行，可以没有前两项

样例：

Const A = 1;

Const char B = ‘b’;

Char C;

C = A + B;

**21.＜参数表＞ ::= ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}**

分析：参数表用于函数参数的传递，可以有多个参数，中间用，逗号分隔

样例：int A或int x, int y

**22.＜主函数＞ ::= void main‘(’‘)’ ‘{’＜复合语句＞‘}’**

分析：定义了主函数的文法，类型为void，同时没有参数但是需要有（）小括号

样例：

Void main(){

Int a, b;

A = 1;

B = 2;

Printf(A + B);

}

**23.＜表达式＞ ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞} //[+|-]只作用于第一个<项>**

分析：该文法定义了表达式的组成，由一个或多个项由加减运算所得，第一个项可以有正负号表示，而后的项不可以，因为是加法运算符，所以优先级低

样例：1或1+2+3或+1+1+1或A+B或+A+B

**24.＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}**

分析：该文法定义了项的构成，项是一个或多个因子相乘所得，因为是乘法运算，优先级高

样例：1\*2\*3或1

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

分析：该文法定义了因子的构成方式，可以是标识符、数组或字符串中的某一项、整数、字符、有返回值的函数调用、或者被（）小括号包裹的表达式

样例：A或B[2]或123或‘C’或add(1,2)或（A+B+C）

**26.＜语句＞ ::= ＜条件语句＞｜＜循环语句＞｜’{‘<语句列>’}‘｜<情况语句>|＜有返回值函数调用语句＞; |＜无返回值函数调用语句＞;｜＜赋值语句＞;｜＜读语句＞;｜＜写语句＞;｜＜空＞;｜＜返回语句＞;**

分析：该文法定义了语句的组成，可以是条件语句、循环语句、情况语句、有返回值的函数调用、无返回值的函数调用或者赋值语句、读写语句、空语句或者返回语句；需要注意分号的使用

（刚刚发现改了语句的文法，可以为大括号包裹的多条语句）

样例：语句的样例由下面不同类型语句显示，在此不做赘述；

{A=A+1;B=B+2}或{{{}}}或{;{;};}

**27.＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞|＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞**

分析：赋值语句可以给标识符赋值、或者数组和字符串的元素赋值，尾部没有分号

样例：A = 1或B[2] = 2 + 3

**28.＜条件语句＞ ::= if ‘(’＜条件＞‘)’＜语句＞else <语句>**

分析：条件语句为if-else，不是语句序列，所以只能有一条语句，不能有大括号；注意没有分号；注意允许嵌套

样例：

if(A == 2) A= A+1;

else A = A – 1;

**29.＜条件＞ ::= ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞ //表达式为0条件为假，否则为真**

分析：该文法用于条件语句、循环语句的判断，既可以是表达式的关系运算，也可以是表达式

样例：A==B或C+D

**30.＜循环语句＞ ::= do＜语句＞while ‘(’＜条件＞‘)’**

分析：循环语句为do-while形式，需要先执行语句，再判断条件；并且不是语句列，只能循环一条语句；注意没有分号

样例：

Do A=A+1

While(A < 100)

**31.＜常量＞ ::= ＜整数＞|＜字符＞**

分析：常量的定义，用于switch语句，为整数或者字符

样例：123或’1’或’A’

**32.＜情况语句＞ ::= switch ‘(’＜表达式＞‘)’ ‘{’＜情况表＞＜缺省＞ ‘}’**

分析：情况语句的定义，switch后面为被（）小括号包裹的表达式，之后是被{}大括号包裹的情况表和缺省

样例：

Switch(A+B){

Case 1: A=A+1;

Case 2: B=B+1;

Default: A=A+B;

}

**33.＜情况表＞ ::= ＜情况子语句＞{＜情况子语句＞}**

分析：情况表用于switch语句，至少有一个情况子句

样例：case 1:A=1+1;或Case 1: A=A+1; Case 2: B=B+1;

**34.＜情况子语句＞ ::= case＜常量＞：＜语句＞**

分析：case后面接常量，同时只能执行一个语句；并且自带break

样例：case 1:A = A+1;

**35.＜缺省＞ ::= default : ＜语句＞**

分析：switch中必然有缺省，缺省后可执行一条语句

样例：default: A=A+B;

**36.＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’|<标识符>** //第一种选择为有参数的情况，第二种选择为无参数的情况

分析：该文法用于有返回值函数的调用，参数依函数定义

样例：add(1,2)或return5

**37.＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’|<标识符>** //第一种选择为有参数的情况，第二种选择为无参数的情况

分析：该文法用于无返回值函数的调用，参数依函数定义

样例：printf5或printf(x)

**38.＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}**

分析：该C0文法的函数调用传参方式为值传递，所以调用的时候实参为表达式，数量依照函数定义

样例：A+B或A,B,C

**39.＜语句列＞ ::=｛＜语句＞｝**

分析：语句列由多个语句构成，可以没有

样例：空或A=A+1或A=A+1；B=B+2；

**40.＜读语句＞ ::= scanf ‘(’＜标识符＞{,＜标识符＞}‘)’**

分析：读语句不能用于读取字符串或数组，只能用于整型或char型标识符；并且读取的类型按照int为%d、char为%c方式进行；可以同时读入多个数据

样例：scanf（a）其中a为整型，意味着读一个整数传给a

或scanf(a,b,c)

**41.＜写语句＞ ::= printf‘(’＜字符串＞,＜表达式＞‘)’|printf ‘(’＜字符串＞‘)’|printf ‘(’＜表达式＞‘)’**

分析：写语句的格式较为奇怪，没有格式化输出，也没有转义，单纯的拼接输出

样例：printf(“a=”,1)，结果输出a=1

或printf（“123”），结果输出123

或printf（1+2+3），结果输出6

**42.＜返回语句＞ ::= return[‘(’＜表达式＞‘)’]**

分析：用于函数的返回，可以没有返回值，直接return；返回值需要有（）包裹

样例：return (x+y)或return或return(0)