## 文法解读

**＜加法运算符＞ ::= +｜-**

作用及限定条件：进行加法或减法运算  
**＜乘法运算符＞  ::= \*｜/**

作用及限定条件：进行乘法或除法运算  
**＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==**

作用及限定条件：进行相等或不等运算，运算结果为boolean型  
**＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z**

作用及限定条件：所有大写字母+所有小写字母+下划线  
**＜数字＞   ::= ０｜＜非零数字＞**

作用及限定条件：数字0-9  
**＜非零数字＞  ::= １｜．．．｜９**

作用及限定条件：数字1-9  
**＜字符＞    ::=  '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'**

作用及限定条件：字母和数字进行加减乘除运算

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

作用及限定条件：由双引号括起来的指定字符 **＜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＞**

作用及限定条件：程序的整体结构

句子示例：

const int a = 10;

int i , j[5];

char ch;

int add (int x) {x=x+1; return x;}

void main() {int num; num=add(3); printf(“n=”,num); return;} **＜常量说明＞ ::=  const＜常量定义＞;{ const＜常量定义＞;}**

作用及限定条件：一个或多个常量说明 **＜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＞}**

作用及限定条件：常量包括整型和字符类型，同一类型可声明一个或多个常量

句子示例：const int a = 1, b = 2; **＜无符号整数＞  ::= ＜非零数字＞｛＜数字＞｝**

作用及限定条件：不允许前导零 **＜整数＞        ::= ［+｜-］＜无符号整数＞｜０**

作用及限定条件：包括正负整数和零 **＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝**

作用及限定条件：标识符必须由字母开头，由字母和数字组成 **＜声明头部＞   ::=  int＜标识符＞ |char＜标识符＞**

作用及限定条件：声明包括整型和字符类型 **＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}**

作用及限定条件：一个或多个变量定义 **＜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＞‘]’ )}**

作用及限定条件：同一类型可定义一个或多个变量，变量包括整数、字符、整数数组和字符数组

句子示例：char c, ch[10]; **＜常量＞   ::=  ＜整数＞|＜字符＞**

作用及限定条件：常量包括整数和字符 **＜类型标识符＞  ::=  int | char**

作用及限定条件：表示整型或字符类型的标识符 **＜有返回值函数定义＞  ::=  ＜声明头部＞‘(’＜参数＞‘)’ ‘{’＜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**

作用及限定条件：有返回值函数分为有参数和无参数两种情况；<声明头部>为返回值类型

句子示例：int add (int x) {x=x+1; return x;}

**＜无返回值函数定义＞  ::= void＜标识符＞(’＜参数＞‘)’‘{’＜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**

作用及限定条件：无返回值函数分为有参数和无参数两种情况；以void开头

句子示例：void add (int x, int y) {x=x+y;}

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

作用及限定条件：复合语句包括常量说明、变量说明和各种语句 **＜参数＞    ::= ＜参数表＞**

作用及限定条件：... **＜参数表＞    ::=  ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}**

作用及限定条件：可定义一到多个参数 **＜主函数＞    ::= void main‘(’‘)’‘{’＜复合语句＞‘}’**

作用及限定条件：主函数的开头 **＜表达式＞    ::= ［+｜-］＜项＞{＜加法运算符＞＜项＞}**

作用及限定条件：可对项进行加法或减法运算 **＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}**

作用及限定条件：对因子进行乘法或除法运算 **＜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＞**

作用及限定条件：注意因子不包括无返回值函数调用语句，也不能为空 **＜语句＞    ::= ＜条件语句＞｜＜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＞;｜＜赋值语句＞;｜＜读语句＞;｜＜写语句＞;｜＜空＞;|＜情况语句＞｜＜返回语句＞;**

作用及限定条件：各类语句；注意包括“空” **＜赋值语句＞   ::=  ＜标识符＞=＜表达式＞|＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞**

作用及限定条件：常量或变量（包括数组）的赋值

句子示例：x = x\*x+100 **＜条件语句＞::= if ‘(’＜条件＞‘)’＜语句＞else＜语句＞**

作用及限定条件：if语句必须包括else

句子示例：if (x>y) return 1; else return -1; **＜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**

作用及限定条件：用于if语句和while语句的条件 **＜循环语句＞   ::=  while ‘(’＜条件＞‘)’＜语句＞**

作用及限定条件：while语句

句子示例：while (x!=10) x=x+1; **＜情况语句＞  ::=  switch ‘(’＜表达式＞‘)’ ‘{’＜情况表＞[＜缺省＞] ‘}’**作用及限定条件：switch语句，包含一或多个case，default可有可无

句子示例：switch (x) {case 1: return 1; case 2: return 2; default: return 0;}

**＜情况表＞   ::=  ＜情况子语句＞{＜情况子语句＞}**

作用及限定条件：可有一或多个分支 **＜情况子语句＞  ::=  case＜常量＞：＜语句＞**

作用及限定条件：case语句的分支 **＜缺省＞   ::=  default : ＜语句＞**

作用及限定条件：case语句的最后一个分支 **＜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**

作用及限定条件：有返回值函数分为有参数和无参数两种情况；<标识符>为函数名

句子示例：add (101, 999) **＜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**

作用及限定条件：无返回值函数分为有参数和无参数两种情况；<标识符>为函数名

句子示例：add (2017) **＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}**

作用及限定条件：传入函数的参数，可有一个或多个 **＜语句列＞   ::= ｛＜语句＞｝**

作用及限定条件：一个或多个语句，或为空 **＜读语句＞    ::=  scanf ‘(’＜标识符＞{,＜标识符＞}‘)’**

作用及限定条件：从输入读取一个或多个字符

句子示例：scanf (value, length) **＜写语句＞    ::= printf ‘(’ ＜字符串＞,＜表达式＞ ‘)’| printf ‘(’＜字符串＞ ‘)’| printf ‘(’＜表达式＞‘)’**

句子示例：printf (sum, “Hello world”)

作用及限定条件：输出字符串或表达式 **＜返回语句＞   ::=  return[‘(’＜表达式＞‘)’]**

作用及限定条件：可返回表达式，也可直接返回

附加说明：

（1）char类型的表达式，用字符的ASCII码对应的整数参加运算，在写语句中输出字符

（2）标识符不区分大小写字母

（3）写语句中的字符串原样输出

（4）情况语句中，switch后面的表达式和case后面的常量只允许出现int和char类型；每个情况子语句执行完毕后，不继续执行后面的情况子语句

（5）数组的下标从0开始