＜加法运算符＞ ::= +｜-

作用：定义了加法运算符有且仅有“+”“-”

限定条件：在加法运算符位置出现其他符号都是违反文法定义

句子示例：+16; a - b

＜乘法运算符＞  ::= \*｜/

作用：定义了乘法运算符有且仅有“\*”“/”

限定条件：在乘法运算符位置出现其他符号都是违反文法定义

句子示例：a \* 10; c / d / e

＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==

作用：定义了两个因子之间的关系有且仅有以上六种

限定条件：出现除以上六种以外的关系运算符是违反文法定义的

句子示例：a <= 10; I >= 0; flag!=0

＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

作用：定义了字母可推导出所有大小写英文字母和“\_”

限定条件：标识符不区分大小写，“\_”指我们并不关注其含义的变量，出现其他字符是错误的

句子示例：a; b; \_; z

＜数字＞   ::= ０｜１｜．．．｜９

作用：数字有且仅有“0,1,…9”

限定条件：数字部分出现其他字符是违反文法定义的

句子示例：0, 1, 9

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

作用：定义了“字符”可以推导出的组成部分

限定条件：字符仅可以推导出“加法运算符”“乘法运算符”“字母”“数字”四种

句子示例：‘+’‘\*’‘a’‘1’

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

作用：定义了“字符串”的组成成分

限定条件：字符串由至少0个至多无穷个ASCII编码符合要求的“字符”组成

句子示例：‘abcdef’‘s686’

＜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：定义了一个程序的结构和可能出现的成分

限定条件：“常量说明”与“变量说明”可以出现一次或不出现；可以出现至少0个至多无穷个有返回值的或无返回值的函数定义；最终一定要有一个主函数

句子示例： const int a = 1;

char c;

foo(0);

foo2(a+b/(c-d));

void main(){return 0;}3

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

作用：定义“常量说明”的格式

限定条件：至少出现一个常量定义，之后可接任意个常量的定义语句

句子示例：const int a = 1;

＜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：定义了常量定义的格式和成分

限定条件：每一句常量定义只能定义int或char中的一种，但是一句可以定义多个同型变量

句子示例：int a = 1, b = 2, c = 3; char d = ‘d’, e = ‘e’;

＜无符号整数＞  ::= ＜数字＞｛＜数字＞｝

作用：定义了无符号整数的生成规则，由至少一个数字至多无穷个组成

限定条件：允许前导零

句子示例：a = 0123456789;

＜整数＞        ::= ［＋｜－］＜无符号整数＞

作用：定义了整数的生成规则

限定条件：必须包含一个“无符号整数”，数字前可有一个加法运算符或什么都没有

句子示例： +16061146; -19491001; 666;

＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝

作用：定义了标识符的命名生成规则

限定条件：必须由字母打头，之后可以接任意个字母或数字

句子示例：a1196116694; adobe; IGNB;

＜声明头部＞   ::=  int＜标识符＞ |char＜标识符＞

作用：定义了“声明头部”的生成规则

限定条件：有两种生成模式，int型与char型

句子示例：int a; char c;

＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}

作用：定义了“变量说明”的生成规则

限定条件：至少有一个“变量定义”，后接任意个变量的定义

句子示例： int a, b[10], c; char d, e[32], f;

＜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

作用：定义了变量定义的生成规则

限定条件：对于每一个类型标识符，至少要有一个声明的主体

句子示例：int a, b[10], c;

＜常量＞   ::=  ＜整数＞|＜字符＞

作用：定义了常量的生成规则

限定条件：常量只有两种类型，整型常量或字符串常量

句子示例：10；‘array\_max\_length’

＜类型标识符＞      ::=  int | char

作用：定义了“类型标识符”的生成规则

限定条件：标识符只有int char

句子示例：int; char;

＜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

作用：定义了有返回值的函数定义的生成规则

限定条件：声明头部与复合语句为必要的成分，参数表为可选项

句子示例：int foo（int a）{a = a + 1; return （a）;}

＜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

作用：定义了无返回值的函数定义的生成规则

限定条件：声明头部与复合语句为必要的成分，参数表为可选项

句子示例：void foo(char c){return ;}

＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞

作用：定义了复合语句的生成规则

限定条件：除去常量说明和变量说明部分，必有的成分为语句列

句子示例：int a = 100； char c = ‘s’; int b;

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

作用：定义了“参数表”的生成规则

限定条件：至少要有一项“＜类型标识符＞＜标识符＞”

句子示例：char strings, int numbers;

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

作用：定义了主函数main的生成规则

限定条件：格式很固定

句子示例：void main(){return （0）;}

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

作用：定义了表达式的生成规则

限定条件：第一处加法运算符为可选项，之后可以有任意次重复

句子示例：+a-10\*(b+c)

＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}

作用：定义了项的生成规则

限定条件：至少有一个因子

句子示例：10 \* (b + c)

＜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：定义了“因子”的生成规则

限定条件：因子可以有很多种类型

句子示例：10；’c’；foo(1); a; a[10];(a+1)

＜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＞;|＜返回语句＞;

作用：定义了“语句”的生成规则

限定条件：语句只可生成以上语句中的一种，或者生成语句列嵌套定义

句子示例： if(i<10) I = I + 1; else I = I – 1;

while(i<10) I = I + 1;

for(i=0;i<10;i=i+1) j = I;

foo(a, b, c+1);

a[1] = c + 10;

scanf(a,b,cc)；

printf(‘scse666’，a+b\*c)；

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

作用：定义了“赋值语句”的生成规则

限定条件：左端的“＜标识符＞＝”为必选项，左侧有两种模式

句子示例：a = 10; a[1] = c + 10;

＜条件语句＞::= if ‘(’＜条件＞‘)’＜语句＞[else＜语句＞]

作用：定义了条件语句（if else）的生成规则

限定条件：else分支是可以省略的，并且不符合以上规则的语句是违反文法的

句子示例：if(i<10) I = I + 1; else I = I – 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：定义了“条件”的生成规则

限定条件：“条件”只有两种模式，且表达式为0条件为假，否则为真

句子示例：if(a); if(a>5)

＜循环语句＞   ::=  while ‘(’＜条件＞‘)’＜语句＞ |for'('＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝

＜标识符＞(+|-)＜步长＞')'＜语句＞

作用：定义了循环语句的生成规则

限定条件：有while和for两种循环模式

句子示例： while(i<10) I = I + 1;

for(i=0;i<10;i=i+1) j = I;

＜步长＞::= ＜无符号整数＞

作用：定义了步长的生成规则

限定条件：步长必须是无符号的整数

句子示例：I = I +1; j = j + 1024;

＜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：定义了“有返回值的函数调用语句”的生成规则

限定条件：有两种模式，其他的形式为违反文法的语句

句子示例：foo(a, b, c+1)

＜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：定义了无返回值函数调用语句的生成规则

限定条件：有两种模式，值参数表为可选项

句子示例：foo(a, b, c+1)

＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}

作用：定义了值参数表的生成规则

限定条件：至少有一个表达式

句子示例：a, b, c+1

＜语句列＞   ::= ｛＜语句＞｝

作用：定义了语句列，用于对语句进行嵌套定义

限定条件：语句列也可能生成一个空语句，即0~无穷个语句

句子示例：while(i<10){I = I + 1; printf(‘success’)}

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

作用：定义了“读语句”的生成规则

限定条件：必备的格式为“scanf(…)”，内可以包含至少一个标识符

句子示例：scanf(a,b,cc)

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

作用：定义了“写语句”的生成规则

限定条件：printf有三种模式，除此以外的格式均为违反文法规则的

句子示例： printf(‘scse666’，a+b\*c)；

printf(‘scse666’)

printf( a+(b\*(c+10))

＜返回语句＞   ::=  return[‘(’＜表达式＞‘)’]

作用：定义了返回语句return的生成规则

限定条件：只有一种正确的表达格式，其余句子均为违反文法表达的非法格式

句子示例：return （a）; return （0）; return（a+(b\*c)）;

附加说明：  
（1）char类型的变量或常量，用字符的ASCII码对应的整数参加运算  
（2）标识符不区分大小写字母  
（3）写语句中，字符串原样输出，单个字符类型的变量或常量输出字符，其他表达式按整型输出   
（4）数组的下标从0开始  
（5）for语句先执行一次循环体中的语句再进行循环变量是否越界的测试