1、＜加法运算符＞ ::= +｜-

范例：c=a+b，d=a-b

分析：加法运算符前后是两个项，进行加减运算，或者在一个项前表示正负

2、＜乘法运算符＞  ::= \*｜/

范例：c=a\*b,d=a/b

分析：乘法运算符前后是两个项，进行乘除运算

3、＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==

范例：if(a!=b){}

if(c==d){}

分析： <表示小于；<=表示小于等于；>表示大于；>=表示大于等于；!=表示不等于；==表示恒等于；这些比较符号多用于条件语句中，作为执行条件

4、＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

示例：char flag::=v;

分析：字母包括小写字母a-z和大写字母A-Z和下划线

5、＜数字＞   ::= ０｜１｜．．．｜９

示例：int line::=90;

分析：数字包括0至9的数字

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

示例：’+’，’0’

分析：字符包括+，-，\*，/，字母和数字，不包括特殊符号

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

示例：chat s=”\*bu6 6\_a^5”

分析：字符串和字符没有确切关系，包含广泛，长度不受限制

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＞

示例：const int a=5,b=7;

const char sll=’v’,sdd=’b’;//常量说明

int c,line[10];//变量说明

int fun(int x,char y){fun分程序(略)}

void sort(int d[5],int e){sort分程序(略)}

int main(){主程序(略)}

分析：常量说明部分与变量说明部分可有可无但顺序不能变，有返回值函数与无返回值函数可以没有可以有多个，且顺序没有要求，主函数必须在最后，且只有一个

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

示例：const int a=5,b=7;

const char sll=’v’,sdd=’b’;

分析：常量说明部分可以有多个const，但每一个const中只包含一种类型int或char，该类型的声明至少有一个

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＞}

范例：int line = 20, cc = 50;

char s = ’k’, flag = ‘ a’;

分析：常量定义只包含一种类型，int或char，每种类型都可以有多个声明

11、＜无符号整数＞  ::= ＜数字＞｛＜数字＞｝

范例：int a = 200398;

分析：无符号整数由0到9之间的字符串组成，没有长度顺序限制

12、＜整数＞        ::= ［＋｜－］＜无符号整数＞

范例：整数在无符号整数的前面加上加号或减号，加号或减号不一定要有

13、＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝

示例：char lost2first = ‘d’;

分析：标识符的第一个字符必须是字母，其后的字符为字母或数字，顺序数量不限

14、＜声明头部＞   ::=  int＜标识符＞ |char＜标识符＞

示例：int fun(char s,int a){}

分析：声明头部用于声明函数的类型和名字，类型有int、char两种，名字为标识符

15、＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}

示例：int add, line[22];

char s, str[45];

分析：变量声明包括至少一个变量定义

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

示例：int add, line[22];

分析：每一个变量定义包含一个int或char，其后为一般变量或数组，数组方括号中必须为无符号整数，一般变量和数组一共至少有一个，声明顺序不限

17、＜类型标识符＞      ::=  int | char

示例：int a = 8;char prinf(int num,char cs){}

分析：类型标识符包括int类型和char类型，用于声明函数或变量、常量的类型

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

示例：int fun(int a, int b){

int add, line[22];

if(a<b){

add=3;

}

else{

add=2;

}

return add;

}

分析：有返回值函数定义首先为声明头部，用于定义返回值的类型，有参数的函数有参数表，无参数的函数无参数表，函数主体为复合语句。

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

示例：void sort{

printf(“there is an error!”);

}

分析：无返回值函数定义开始必须为void<标识符>，有参数的函数有参数表，无参数的函数无参数表，函数主体为复合语句。

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

示例：const int a=5,b=7;

int add, line[22];

if(a<b){

add=3;

}

分析：复合语句中常量说明和变量说明可有可无，但顺序不可变，其后为语句列

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

示例：int fun(int a, char b){}

分析：参数表用于有参数的函数，参数由函数外部输入，进入函数参与运算，函数内部对参数的改变会影响外部

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

示例：void main(){

Printf(“Hello Word!”);

}

分析：主函数不返回值，开头为void，由于为最外层的函数，括号内没有参数，函数主体为复合语句。

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

示例：-30+ fun(int a,int b)

分析：表达式至少包含一个项，相邻两项间必须有加法运算符+或-，第一个项前面可以有+或-表示正负，也可以没有默认无符号

24、＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}

示例：line \* sd

分析：项由至少一个因子组成，相邻因子间必须有乘法运算符\*或/

25、＜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＞

示例：line[2]

fun(int a,int b)

分析：因子可以为标识符，标识符必须有值，可以为数组元素，可以为(表达式)，可以为整数，可以为字符，计算时转为assic码，可以为有返回值调用语句，因子必须能够计算出值，能够用于计算

26、＜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＞;

示例：while(a){

printf(“ad”);

a=a-1;

}

分析：语句包含条件语句、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、空语句、返回语句，只含有一个上述成分

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

示例：first=a+b;

line[2]=c+d;

分析：赋值语句等号前面为标识符或数组元素，等号后的表达式的值赋给该成分

28、＜条件语句＞::= if ‘(’＜条件＞‘)’＜语句＞[else＜语句＞]

示例：if(a==b){

printf(“true”);

}

else{

printf(“false”);

}

分析：条件语句必须有if和紧跟其后的语句成分，else及其后的语句成分可有可无

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

示例：a+b<=c

分析：条件为一个表达式或两个表达式和其之间的关系运算符，表达式可以计算出值来进行比较，关系运算符为<｜<=｜>｜>=｜!=｜==，用于比较得出true或flase，进行条件判断

30、＜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＞  
 示例：for(a=3; a<10;a=a+1){

printf(a);

}

分析：循环语句包括while循环和for循环，while循环的括号内为条件，主体为语句，for循环括号内有三部分，第一部分为＜标识符＞＝＜表达式＞，用于给标识符赋初值，第二部分为条件，判断是否结束循环，第三部分为＜标识符＞＝＜标识符＞(+|-)＜步长＞，改变表示符的值，进行下一次循环和判断

31、＜步长＞::= ＜无符号整数＞

示例：6

分析：步长用于每次循环结束后对标识符进行改变

32、＜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

示例：add(a+b, c);

分析：有返回值函数调用语句用于调用有返回值函数，得到返回值，有参数时括号内为传入函数并进行运算的参数

33、＜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

示例：fun(char c)；

分析；无返回值函数调用语句用于调用无返回值函数，进行相关操作，有参数时括号内为传入函数并进行运算的参数

34、＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}

示例：a+b,c

分析：值参数表至少包含一个表达式，相邻表达式间有逗号隔开，每个表达式都是一个参数，传入函数进行相关运算。

35、＜语句列＞   ::= ｛＜语句＞｝

示例：add(a+b, c);

fun(char c)；

for(a=3; a<10;a=a+1){

printf(a);

}

分析：语句列包含任意多个语句，可以为空

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

示例：scanf(a,b,c);

分析：读语句读取外部输入的值并传递给变量

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

示例：printf(“hello world”,a+b);

分析：写语句用于输出字符换或表达式的值，每个写语句最多包含一个字符串和一个表达式，且至少包含其中之一，字符串和表达式同时存在时，顺序不能变

38、＜返回语句＞   ::=  return[‘(’＜表达式＞‘)’]

示例：return a;

分析：返回语句用于返回函数值或跳出循环，没有表达式时，不返回，跳出循环或函数