1. ＜加法运算符＞ ::= +｜-

作用:定义加减运算符号

限定条件:表达式计算

Ex:a=1+1,c=a-b

2. ＜乘法运算符＞ ::= \*｜/

作用:定义乘除运算符号

限定条件:表达式计算

Ex:a=2\*3,c=a/b

1. ＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==

作用:定义关系运算符小于、小于等于、大于、大于等于、等于、不等于

限定条件:用于关系运算,条件判断

Ex:while(a==1)

4. ＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

作用:定义了字母集合

限定条件: 可用于标识符命名，字符串表示，字符常量表示等

Ex:a,A,m

5. ＜数字＞ ::= ０｜１｜．．．｜９

作用:定义了数字的集合

限定条件: 可用于表示数字,也可表示逻辑值

Ex:1,3,5,7,9

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

作用:定义了单字符

限定条件:单字符只能由此产生

Ex:+,-,\*,/,a,1

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

作用:字符串组成文法

限定条件:字符串中字符只能是十进制编码为32,33,35-126的ASCII字符

Ex:”abc45”

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＞

作用:定义了程序的大体结构

限定条件:非主函数必须在主函数之前进行定义,取消了C的函数声明

Ex:const int a = 1;

int c;

int emm(int rua){return rua;}

void fu(){c = a}

void main(){emm(2);}

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

作用:定义常量的格式

限定条件:必须以const开头

Ex:const int a = 1;

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＞}

作用:常量定义的组成

限定条件:常量只能为int和char类型

Ex:int a = 1;

11. ＜无符号整数＞ ::= ＜数字＞｛＜数字＞｝

作用:定义了无符号整数的组成

限定条件:支持前导0

Ex:001,2345

12. ＜整数＞ ::= ［＋｜－］＜无符号整数＞

作用:定义了整数组成

限制条件:可选择加+-号

Ex:+001,-02543,555

13. ＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

作用:定义了标识符的组成

限制条件:以英文字母或者下划线开头,可接数字字母和下划线

Ex:a\_01sd

14. ＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞

作用:声明头部组成,用于有返回值函数定义

限制条件:只能是int和char类型

Ex:int A

15. ＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}

作用:变量说明组成

限制条件:全局声明或复合语句中声明

Ex:int a,b,c

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

作用:变量定义组成

限制条件:至少定义一个,以类型标识符开头

Ex:int a,b[5]

17. ＜类型标识符＞ ::= int | char

作用:定义支持的类型标识符

限定条件:只能为int和char

Ex:int

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

作用:有返回值函数结构

限定条件:以声明头部开始,必须有函数体

Ex:int A{return 1;}

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

作用:无返回值函数结构

限定条件:必须以void开头

Ex:void A(int a){a = 1;return;}

20. ＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

作用:复合语句结构

限定条件:只能用于函数体内

Ex:const int a = 1;int b;b = a \* a;

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

作用:参数表结构

限定条件:不能为空,参数必须为类型标识符定义的类型

Ex:int a,char b

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

作用:主函数结构组成

限定条件:主函数是void类型,无参数

Ex:void main(){print(“1”);}

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

作用:表达式组成

限定条件:只能推导出加减项

Ex:-7+8-9

24. ＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}

作用:项的结构组成

限定条件:只能推导出乘除因子

Ex:3\*5

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＞‘)’

作用:因子组成部分

限定条件: 因子可以是变量、数组某个元素、某个表达式、带小括号的某个表达式、整数、单字符(ascii值)、有返回值的函数调用

Ex:a a[0] ‘a’ a\*[a+1] (4+7)

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＞;

作用:语句组成部分

限定条件:只能在函数体内

Ex:a = b + 1;

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

作用:赋值语句组成部分

限定条件:用于赋值,末尾不带;

Ex:a = b + 1

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

作用:条件语句组成部分

限定条件:末尾没有;,if和else必须都有

Ex:if(x==1){x = 2;}else{x = 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

作用:定义了条件语句的结构

限定条件:0为false,其余为true

Ex:a>=1

30. ＜循环语句＞ ::= do＜语句＞while ‘(’＜条件＞‘)’

作用:定义了do while的结构

限定条件:只支持do while

Ex:do{a = a + 1;}while(a < 6)

31. ＜常量＞ ::= ＜整数＞|＜字符＞

作用:常量的组成

限定条件:整数或者单字符

Ex:154 , A

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

作用:条件选择语句的组成结构

限定条件:none

Ex:int a;

a = 1;

switch(a){

case1:a = 2;

case2:a = 3;

default: a = 1;

}

33. ＜情况表＞ ::= ＜情况子语句＞{＜情况子语句＞}

作用:情况表组成部分

限定条件:只能在情况语句中

Ex:如32的case部分

34. ＜情况子语句＞ ::= case＜常量＞：＜语句＞

作用: 情况子语句组成部分

限定条件:case开头,case后跟常量

Ex: 如32的case部分

35. ＜缺省＞ ::= default : ＜语句＞

作用:缺省情况结构

限定条件:只出现在情况语句中

Ex:如32

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

作用:有返回值函数调用结构

限定条件:无参数不加括号

Ex:Fib(10)

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

作用:无返回值函数调用结构

限定条件无参数不加括号:

Ex:exit(0)

38. ＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}

作用:值参数表结构

限定条件:只能为表达式

Ex:a + b

39. ＜语句列＞ ::=｛＜语句＞｝

作用:语句列结构

限定条件:语句列可为空

Ex: if(x==1){x = 2;}else{x = 1;}

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

作用:读语句组成

限定条件:只能处理int和char

Ex:scanf(a)

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

作用:写语句组成部分

限定条件:以printf开头

Ex:printf(“asdasd”) , printf(7\*7\*7\*7\*7)

42. ＜返回语句＞ ::= return[‘(’＜表达式＞‘)’]

作用:返回语句组成

限定条件:以return开头

Ex:return , return(a+d)