1. ＜加法运算符＞ ::= +｜-

作用:定义了运算符号加与减

限定条件：用于表达式计算

ex: a = 3+2; b = a-b;

1. ＜乘法运算符＞ ::= \*｜/

作用:定义了运算符号乘和除

限定条件：用于表达式计算

ex:a = 4\*5; b = a/2;

1. ＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==

作用：定义关系比较运算符小于、小于等于、大于、大于等于、等于、不等于

限定条件：用于关系比较运算,产生条件判断的依据bool值

ex:if(a>8) x = x+1; else x = x\*3;

4. ＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

作用：定义了字母集合

限定条件：用于标识符命名，字符串表示，字符常量表示等

ex：int,char,\_linux

1. ＜数字＞ ::= ０｜＜非零数字＞

作用：定义了单数字集合0，1，2，3，4，5，6，7，8，9

限定条件：用于数字串的组合表示，也可表示逻辑判断值

ex：0，1，2，3，4，5，6，7，8，9

1. ＜非零数字＞ ::= １｜．．．｜９

作用：定义了单非零数字集合

限定条件：用于数字串的组合表示，也可表示逻辑值

ex：1，2，3，4，5，6，7，8，9

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

作用：定义单字符

限定条件：所有的单字符常量只能由此文法产生

ex：’+’,’-’,’a’,’b’,’3’

8. ＜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文法

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

ex：”abc”,”kiln”;

9. ＜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：定义了整个C0文法支持的程序代码的组成结构

限定条件：程序从前往后(自订向下)组成为：全局常量说明部分(可无)、全局变量说明部分(可无)、0或多个有返回值的函数定义或者无返回值的函数定义、主函数，非主函数的定义必须放在主函数前，即取消了C语言的函数申明。

ex: const int a = 13; const char b = ‘c’;

int x,y; char p,q;

int lala(int bbq){return bbq+bbq;}

void kaka(char ppt){p = ppt;}

void main(){

int ttt,ppp;

char lll,aaa;

ttt = lala(x+y);

}

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

作用：定义了常量的组成成分

限定条件：常量申明必须以const开头,以常量的定义部分结尾。

ex:const int a = 23;

11. ＜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：定义了常量定义具体组成。

限定条件：常量只能是int型和char型的常量

ex: const int a = 23,b=56; const char x = ‘c’,y=’d’;

12. ＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝

作用：定义了无符号整数，用于数组声明的长度信息

限定条件：0不在无符号整数定义文法内,不存在前导0

ex：23，1，4445

13. ＜整数＞ ::= ［＋｜－］＜无符号整数＞｜０

作用：定义了所有的整数

限定条件：不存在前导0(整数0本身不算是前导0)

ex:0,+123,123,-123

14. ＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

作用：定义了标识符的组成成分

限定条件：标识符必须以英文字母或者下划线开头,组成部分只能是字母下划线或者数字。

ex:\_buaa,Apple,linux,Buaa1506

15. ＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞

作用：定义了声明头部的组成部分,用于有返回值的函数定义

限定条件：有返回值的函数的返回值只能是int或者char类型的

ex：int kaka{return 23;}

16. ＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}

作用：定义了变量说明的组成

限定条件：用于全局变量声明和复合语句中的变量声明

ex：char a,b,c,d;int ddd;

17. ＜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：定义了变量定义的组成内容

限定条件：必须以类型标识符开头，一次可以声明多个同类型的变量。

ex：int x,y[3];

18. ＜常量＞ ::= ＜整数＞|＜字符＞

作用：定义了常量数值的组成部分

限定条件：常量只能是整数或者单字符

ex:-34 -45 ‘c’ ‘+’

19. ＜类型标识符＞ ::= int | char

作用：定义了程序支持的类型标识符有哪些

限定条件：只能是int或者char

ex:None

20. ＜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：定义了有返回值的函数定义的文法

限定条件：必须有函数头部,有参数需要左右小括号,无参数可以不写小括号,函数体必须以{}包裹

ex:int xx(int b){return b+b;}

21. ＜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：定义了无返回值的函数定义的文法

限定条件：必须以void开头,有参数需要左右小括号,无参数可以不写小括号,函数体必须以{}包裹

ex:void xx(int b){b=23;return;}

22. ＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

作用：定义了复合语句的组成结构,是函数程序体的全部

限定条件：适用于函数体,函数体外不可用

ex: const int a = 23; int x; x = a;

23. ＜参数＞ ::= ＜参数表＞

作用：定义参数的组成，实际参数可直接用参数表文法替换

限定条件：无

ex: void xx(int b){b=23;return;}中的int b就是参数

24. ＜参数表＞ ::= ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}| ＜空＞

作用：定义了参数表的组成成分

限定条件：参数表可为空串，如果不为空，每个参数都必须是类型标识符定义范围内的类型数据

ex: ex: void xx(int b,char c){b=23;return;}中的int b,char c就是参数表内容

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

作用：定义了主函数的组成成分

限定条件：必须以void main开头

ex:void main(){printf(“Hello World.”);}

26. ＜表达式＞ ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}

作用：定义了表达式的组成成分

限定条件：只能用于求值计算过程

ex: -5+6+a[4]

27. ＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}

作用：定义了项的组成部分

限定条件：是表达式的组成部分

ex:3\*5

28. ＜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 b[4] ‘c’ 4\*5 4+8

29. ＜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：定义了语句的组成成分

限定条件：用于函数体内

ex:if(x==1) printf(“x==1”);else printf(“x!=1”);

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

作用：定义了赋值语句的组成成分

限定条件：用于表达式计算

ex: a = b[2]

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

作用：定义了条件转移语句的成分

限定条件：<条件的最终值必须是逻辑相关的值>

ex:if(x>=10) printf(“大于等于10”);else printf(“小于10”);

32. ＜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：定义了条件语句中的判断条件的具体组成

限定条件：条件最终对应一个整数结果(0为false,其他为true)

ex:a>=10 ; 5-5;

33. ＜循环语句＞ ::= while ‘(’＜条件＞‘)’＜语句＞

作用：定义了循环语句的结构

限定条件：只支持while循环

ex:while(x>=10){x--;printf(“Hello World”);}

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

作用：定义条件选择语句(情况语句)的组成成分

限定条件:无

ex: int a;

a = 10;

switch(a){

case 10:printf(“ten”);

case 9 : printf(“nine”);

default:printf(“What!”);

}

35. ＜情况表＞ ::= ＜情况子语句＞{＜情况子语句＞}

作用：定义了情况表的组成成分

限定条件：只可以用于情况语句中

ex:上题example中的case语句集合

36. ＜情况子语句＞ ::= case＜常量＞：＜语句＞

作用：定义了情况子语句的组成

限定条件：必须以case开头,选择依据的值必须是常量

ex: case 9 : printf(“nine”);

37. ＜缺省＞ ::= default : ＜语句＞

作用：定义了情况语句中缺省情况的组成

限定条件：只可以用于情况语句中

ex:default : printf(“DEFAULT”);

38. ＜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：定义有返回值的函数调用的具体组成

限定条件：对于无参数的函数调用需要注意不能加小括号

ex: x = Fibonaci(6);

39. ＜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：定义无返回值的函数调用的具体组成

限定条件：对于无参数的函数调用需要注意不能加小括号

ex: exit(1);

40. ＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}

作用：定义值参数表的组成成分

限定条件：只能是表达式

ex: 23，a+b

41. ＜语句列＞ ::= ｛＜语句＞｝

作用：定义了语句组成

限定条件：组成成分为语句,可以为空

ex: {printf(“x”); printf(“y”);printf(“z”);}

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

作用：定义读语句的组成成分

限定条件：只能处理int和char，必须以scanf开头

ex:scanf(x,y,z);

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

作用：定义写语句的组成部分

限定条件：以printf开头

ex: printf(“x”,6+8); printf(“y”);printf(3+6);

44. ＜返回语句＞ ::= return[‘(’＜表达式＞‘)’]

作用：定义返回语句的组成

限定条件：必须以return开头，表达式左右必须有括号

ex:return (a+b);