1.

＜加法运算符＞ ::= +｜-

eg:

略

analysis:

略

2.

＜乘法运算符＞ ::= \*｜/

eg:

略

analysis:

略

3.

＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==

eg:

略

analysis:

略

4.

＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

eg:

略

analysis:

略

5.

＜数字＞ ::= ０｜＜非零数字＞

eg:

略

analysis:

略

6.

＜非零数字＞ ::= １｜．．．｜９

eg:

略

analysis:

略

7.

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

eg:

略

analysis:

略

8.

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

eg:

dfshjsdfhjksdfFFFDFDSii！ ;456sdfsd546

analysis:

包括的符号有“ !#$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~”，需要注意的是不包括双引号

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＞

eg:

const int constnumber = 1; //［＜常量说明＞］

int varnumber; //［＜变量说明＞］

void print(){ // ＜有返回值函数定义＞

printf("\n");

}

void main(){ // ＜主函数＞

print();

}

analysis:

最多只能有一个块用来声明常量，一个块用来声明变量。 可以没有或有一个或多个有无返回值的函数定义。

并且这三个快的顺序固定。

10.

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

eg:

const int constnumber0 = 0;

const int constnumber1 = 1;

analysis:

可以有多个或者一个常量的定义。每定义前是const，后面是“;”。

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＞}

eg:

const int constnumber0 = 10, constnumber1 = 11, constnumber2 = 12;

const char constchar0 = 'd', char constchar1 = 'd', char constchar2 = 'd';

analysis:

const分为int和char两种，每一条常量定义可以定义一种多个常量，每个之间逗号间隔。

12.

＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝｜０

eg\_0:

0

eg\_1:

50

analysis:

无符号整数无前导0。

13.

＜整数＞ ::= ［＋｜－］＜无符号整数＞

eg:

+5

analysis:

整数可以有或无符号，最多一个。

14.

＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

eg:

wangkexiang666

analysis:

标识符只包括字母或者数字，第一个必须是字母。

15.

＜声明头部＞ ::= int＜标识符＞|char＜标识符＞

eg:

int getnumber

analysis:

函数声明头部只能是int或者char，且必须有且仅有一个。

16.

＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}

eg:

int declareint;

char declarechar;

analysis:

至少有一个变量定义，每个以分号结尾。

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＞']') } //＜无符号整数＞表示数组元素的个数，其值需大于0

eg:

int declareint, declareIntArray0[8];

char declarechar, declareCharArray0[9];

int declareIntArray[5];

char declareCharArray[6];

analysis:

一条定义中可以同时定义数组和单个变量。

18.

＜类型标识符＞ ::= int | char

eg:

int

analysis:

类型只包括int和char。

19.

＜有返回值函数定义＞ ::= ＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'

eg:

int getnumber(int n){

return 0;

}

analysis:

＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'三个部分四个符号缺一不可，顺序固定。

20.

＜无返回值函数定义＞ ::= void＜标识符＞'('＜参数表＞')''{'＜复合语句＞'}'

eg:

void operate(int n){

n = n + 1;

}

analysis:

void＜标识符＞'('＜参数表＞')''{'＜复合语句＞'}'一个标识符三个部分四个符号缺一不可，顺序固定。

21.

＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

eg:

const int constInt = 0; // ＜常量说明＞

int intint; // ＜变量说明＞

intint = 0; // ＜语句列＞

intint = 1;

analysis:

最多只能有一个块用来声明常量，一个块用来声明变量。 可以没有或有一个或多个语句。

并且这三个快的顺序固定。

22.

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

eg\_0:

eg\_1:

int n

eg\_2:

int a, int b

eg\_3:

iut a, char b

analysis:

可以有或没有参数，参数只能是int或者char

23.

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

eg:

void main(){

int n;

}

analysis:

只能是void main(){}这样，且返回值只能是void。

24.

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

eg\_0:

+a-b+a\*b

eg\_1:

a-b+a\*b

analysis:

开头可以有一个或没有符号

25.

＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}

eg:

a\*b/c

analysis:

第一个一定是一个因子后面可以加其他因子，之间用乘除相连。

26.

＜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＞')'

eg\_0:

a[b]

eg\_1:

5

eg\_2:

d

eg\_3:

add(a,b)

eg\_4:

(a+b\*c)

analysis:

因子没有符号

27.

＜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＞;

eg\_0:

if(a == 0){

return a;

} else{

return 0;

}

eg\_1:

for(i = 0;i < 10; i = i + 1){

a = a + 1;

}

eg\_2:

{

i = i + 1;

}

eg\_3:

add(a, b);

eg\_4:

merge(a, b, c);

eg\_5:

a = b;

eg\_6:

scanf(a, b);

eg\_7:

printf("%d", a);

eg\_8:

;

eg\_9:

return (0);

eg\_10:

return ;

analysis:

条件语句和循环语句没有分号，返回语句可以直接热return ;

28.

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

eg:

a = b;

c[4] = a\*b;

analysis:

只可以给变量或者数组的某个值赋值一个表达式。

29.

＜条件语句＞ ::= if '('＜条件＞')'＜语句＞［else＜语句＞］

eg:

analysis:

可以没有else的处理

30.

＜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

eg\_0:

a==b

eg\_1:

a

analysis:

可以是一个判罚关系的表达式，也可以判断一个值是不是0。

31.

＜循环语句＞ ::= do＜语句＞while '('＜条件＞')' |for'('＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝＜标识符＞(+|-)＜步长＞')'＜语句＞

eg\_0:

do{

i = i + 1;

} while(i < 10)

eg\_1:

for(i = 0;i < 10; i = i + 1){

a = a + 1;

}

analysis:

对于此文法，for内的三个块必须都存在，不为空。

32.

＜步长＞::= ＜无符号整数＞

eg\_0:

0

eg\_1:

50

analysis:

这里不支持为char的情况

33.

＜有返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'

eg:

add(a\*c, b)

analysis:

略

34.

＜无返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'

eg:

merge(a, b)

analysis:

略

35.

＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}｜＜空＞

eg:

a\*b,c

analysis:

可以有多个或者一个或者没有，每两个逗号间隔

36.

＜语句列＞ ::=｛＜语句＞｝

eg:

a = a + 1;

b = a;

analysis:

语句列甚至可以没有语句。

37.

＜读语句＞ ::= scanf '('＜标识符＞{,＜标识符＞}')'

eg:

scanf(a,b);

analysis:

和一般的不一样，功能缩减。

38.

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

eg:

printf("%d", a)

analysis:

功能有所缩减。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

39.

＜返回语句＞ ::= return['('＜表达式＞')']

eg:

return (0)

analysis:

必须有括号要不然不返回任何值