**14061176 黄秋宇 文法解读**

扩充C0文法-高

* + - * ＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝
* ＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z
* ＜数字＞ ::= ０｜＜非零数字＞
* ＜非零数字＞ ::= １｜．．．｜９
* 分析：标识符必须以字母a-z或下划线开头、由字母或数字或下划线构成。
* 范例：\_1406

\_14compiler

\_14com0pil6er

\_ComPiler

\_co1mpi40ler6

\_

\_\_

\_de\_23\_FEWe

c1406

c14om06pi11ler

c\_e3\_32\_fe

以上均为合法标识符

以下均为非法标识符

23com //不能以数字开头

2\_de3 //不能以数字开头

0com\_ //不能以数字开头

\_com&&de //不能出现非法字符

com%$de //不能出现非法字符

de#@d#2cd //不能出现非法字符

fr\*de&cs^cd //不能出现非法字符

* ＜整数＞ ::= ［＋｜－］＜无符号整数＞｜０
* ＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝
* 分析：整数由零或可带符号的整数组成，即零前面不能出现符号，有符号整数中的符号只能出现一次，且有符号整数没有前导零。
* 范例：0

+12

-23

200

+200

-200

以上均为合法整数

以下均为非法整数

+0 //0前面没有符号

-0 //0前面没有符号

01 //无符号时没有前导零

+01 //有符号时也没有前导零

-01 //有符号时也没有前导零

++1 //符号最多只能出现一次

+-1 //符号最多只能出现一次

--1 //符号最多只能出现一次

-+1 //符号最多只能出现一次

+++++1 //符号最多只能出现一次

00023 //无符号时没有前导零

23+ //数字后面不能再接符号

23- //数字后面不能再接符号

23&&23 //不能出现非法字符

* ＜字符＞ ::= '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'
* ＜加法运算符＞ ::= +｜-
* ＜乘法运算符＞ ::= \*｜/
* 分析：字符由加法运算符或乘法运算符或字母或数字组成，其余均为非法字符，且字符只由单引号加单个字符构成。
* 范例：’+’

‘-‘

‘\*’

‘/’

‘a’

‘z’

‘A’

‘Z’

‘0’

‘1’

‘9’

以上均为合法字符

以下均为非法字符

& //不能出现非法字符

a //未加上单引号

++ //字符是以单字符形式出现

-- //字符是以单字符形式出现

ab //字符是以单字符形式出现

12 //字符是以单字符形式出现

+\* //字符是以单字符形式出现

++1 //字符是以单字符形式出现

%# //字符是以单字符形式出现

* ＜表达式＞ ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}
* ＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}
* ＜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＞‘)’
* ＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’
* ＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}｜＜空＞
* 分析：此处表述了表达式由项组成的具体方式，包含多种情况，需要注意如果参与运算的因子为字符则按照该字符对应的ASCII码对应的整数参加运算。有返回值函数调用中参数可能是数组，数组中的下标也有可能是函数调用的返回值，也有可能不止两层嵌套，这里存在递归定义。
* 范例：+\_com

-\_com

\_com

+com

-com

+c

-c

+com[as]

s[fun(10,ss[10],k)]

s[func(u,i,j)+1]+s[10]

10+10\*10

-10+-19\*10

10\*-10+-19

10+19\*-10

10\*10+10

-com[2]

+c+c

d+wd

cn-10

10-cn

++10

+’+’+10 //第一个+是指表达式的FIRST集合中的，第二个+为项->因子->字符，第三个+为加法运算符

+’+’+’+’

+(++10)

+(-10)

+-10

--10

-(--10)

(12)

(-‘c’)

(-10)

‘a’

‘z’

‘A’

‘Z’

0

‘1’

9

+0 //在这里是合法的

-0 //在这里是合法的

+function(a,n)

function(ooo,2)

function(func1(12,i,j,n),com[k],s[func2(10,0)])

以上均为合法表达式

以下均为非法表达式

function(a,n) //其中function为无返回值函数调用

+++1

+++++++10 //表达式嵌套表达式需要加上括号()

* ＜语句列＞ ::=｛＜语句＞｝
* ＜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＞;
* ＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞|＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞
* ＜条件语句＞ ::= if ‘(’＜条件＞‘)’＜语句＞［else＜语句＞］
* ＜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
* ＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==
* ＜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＞
* ＜步长＞::= ＜无符号整数＞
* ＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’
* ＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’
* ＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}｜＜空＞
* ＜读语句＞ ::= scanf ‘(’＜标识符＞{,＜标识符＞}‘)’
* ＜写语句＞ ::= printf‘(’＜字符串＞,＜表达式＞‘)’|printf ‘(’＜字符串＞‘)’|printf ‘(’＜表达式＞‘)’
* ＜字符串＞ ::= "｛十进制编码为32,33,35-126的ASCII字符｝"
* ＜返回语句＞ ::= return[‘(’＜表达式＞‘)’]
* 分析：此部分主要说明了各种类型语句的结构组成，即说明了能够接受的语句语法格式，某些语句会有多种形式，且存在嵌套。
* 范例：\_com=1+1;

\_com[10]=func(10,n);

i=+1; //语句列中的赋值语句

if(1>a) s=10;

if(s==1) n=1; else n=2;

if(s==1) n=1; else if(s==2) n=2;

if(s==1) ; else ; //语句列中的条件语句（语句可为空）

do a=1; while(a>0)

do ; while(a>0)

for(a=1;n<10;n=n+1) a=a+1;

for(a=1;n<10;n=n+1); //语句列中的循环语句（语句可为空）

func(10,n);

func(); //语句列中的函数调用语句

scanf(a);

scanf(a,b); //语句列中的读语句

printf(“abc”,a=1);

printf(“abc”);

printf(a=1); //语句列中的写语句

return;

return(0); //语句列中的返回语句

{ if(s==1) for(i=1;i<10;i=i+1) {scanf(a);printf(a);}; else n=2;} //语句列嵌套

; //语句列中的空语句

以上为合法语句列

以下为非法语句列

s=1+1 //语句列中的赋值语句后面带分号

s=[10]=func(10,n) //语句列中的赋值语句后面带分号

=1; //赋值语句必须以标识符开始

i+=1; //非法赋值语句

if(a>1) s=10 //条件语句，语句列中语句后面带分号

if(s==1) n=1; else n=2 //条件语句，语句列中语句后面带分号

else n=1; //条件语句必须以if开始

do a=1 while(a>0) //循环语句。do后面的语句须加分号

do a=1; while(a>0); //循环语句。while后面的语句不加分号

do ; while(a>0); //循环语句。while后面的语句不加分号

for(a=1,n=10;n<20;n=n+1) a=a+1; //循环语句。for中初始时只能有一个＜标识符＞＝＜表达式＞，不能出现两个及以上

for(a=1;n<20&&a<10;n=n+1) a=a+1; //循环语句。for中条件判断没有&&

for(a=1;n<20;n=n+1, a=a+1) c=c+1; //循环语句。for的循环增量不能有两个及以上

for(a=1;n<20;n=n+1;) c=c+1; //循环语句。for的循环增量后面不能接分号

for(n=1;n<20;n=n+1) //循环语句。for中的语句后面须接分号

for(n=1;n<20;n=n+0) //循环语句。for中的循环增量不能为0

for(n=1;n<20;n=1+n) //循环语句。for中的循环增量标识符必须在前，增量在后

func(10,n) //函数调用语句后面接分号

func() //函数调用语句后面接分号

func(10,n); //函数声明为func(int j, int k, char c),函数调用语句与函数声明不符

scanf(a) //读语句后面接分号

scanf(1a); //读语句中的标识符非法

printf(“””); //写语句中的字符串不包含双引号”

printf(“) //写语句后面接分号

return //返回语句后面接分号

return 0; //返回语句中的表达式须带括号()

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

* ＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}
* ＜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＞‘]’) }
* ＜类型标识符＞ ::= int | char
* ＜有返回值函数定义＞ ::= ＜声明头部＞‘(’＜参数＞‘)’ ‘{’＜复合语句＞‘}’
* ＜声明头部＞ ::= int＜标识符＞|char＜标识符＞
* ＜无返回值函数定义＞ ::= void＜标识符＞‘(’＜参数＞‘)’‘{’＜复合语句＞‘}’
* ＜主函数＞ ::= void main‘(’‘)’ ‘{’＜复合语句＞‘}’
* ＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞
* ＜参数＞ ::= ＜参数表＞
* ＜参数表＞ ::= ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}| ＜空＞
* 分析：根据文法可知，程序的各个组成成分的声明顺序都已经被限定好了，不能随便更改声明顺序。每一个程序按顺序由常量说明部分、变量说明部分、函数定义及主函数组成。其中复合语句内也有严格的先后顺序声明。程序中函数定义可出现多次。常量说明和变量说明对于程序来说均为可有可无的部分。
* 范例：const int a=1; void main(){;}

const int a=1,b=2; void main(){;} //多个常量定义

void main(){} //语句列中包含0个语句

const char b=’c’; void main(){;}

const char b=’c’;char n,a,m; void main(){;} //多个变量定义

const char b=’c’;char n; char a; int m; void main(){;} //多个变量定义

const char b=’c’;char n,a,m; int func(){;} void main(){;}

const char b=’c’;char n,a,m; int func(){} void main(){;}

int func1(){;} void func2(){;} void main(){;} //多个函数定义

以上均为合法程序

以下均为非法程序

int n[0]

const a=1; void main(){;} //常量定义缺少类型标识符

const int a=1; //缺少主函数

const char b=c; void main(){;} //常量定义字符带单引号

char n; const char b=’c’; void main(){;} //常量说明和变量说明顺序颠倒

const char b=’c’;char n=a; void main(){;} //变量说明不能赋值

const char b=’c’;char n;a; void main(){;} //多个变量之间不能以分号隔开，要以逗号隔开