文法解读

1.＜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＞'('＜参数表＞')' '{'＜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＞'('＜参数表＞')''{'＜复合语句＞'}'

＜主函数＞ ::= void main'('')''{'＜复合语句＞'}'

范例：

const int a=1,b=2;

const char c=’2’;

const int d=33; /\*常量说明\*/

int k,ary[5];

char cc,s[5];

int n; /\*变量说明\*/

int func1(参数表){

const int c=1;

const char cw=’a’;

char ss；

...<复合语句>

}

void func2(参数表){

...<复合语句>

}

int func3(参数表){

const ...

...<复合语句>

}

void main(){

int ...

char ...

...<复合语句>

}

分析1：程序按照一定顺序由常量说明、变量说明、函数定义、主函数构成，而且几部分顺序不能改变，例如“var a；const b=10;”不合法；

“const b=10；var a；

int func1(略){ 略 }

void main(){ 略 }

void func2(略){ 略 }”这也是不合法的；

程序的常量说明、变量说明、函数定义三部分是可有可无的，但是主函数必须有；

函数定义部分可以有多个函数定义，而且有返回值函数和无返回值函数的顺序不做限制；

分析2：根据常量说明的文法：“const int a=1，int c=2；const char b=’k’；”这种常量说明写法是错误的，但是相同类型常量可以共用一个’const’和类型标识符，例如：“const int a=1，b=2；”符合规则，注意用逗号隔开两个常量；

根据常量定义：每个常量声明后必须用等号赋值，“const int a；”这是错误的

分析3：变量定义可以定义数组，定义数组必须定义数组元素个数如“int a[10];”,相同类型变量可以共用一个类型标识符如“int a，b，c[5];”;

根据变量定义，每个变量声明时不能赋值“int a=0；”不合法，如果需要赋值就必须使用赋值语句；

分析4：有返回值函数定义声明头部用于指定返回值类型（int 或 char）以及函数名；无返回值函数需要有void标识符，void后的标识符是函数名；void main()表示主函数声明；

分析5：对于数组定义，该文法没有对于多维数组的定义,’int a[5][5];’不合法；

2.＜数字＞ ::= 0｜＜非零数字＞

＜非零数字＞ ::= １｜．．．｜９

＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝| 0

＜整数＞ ::= ［＋｜－］＜无符号整数＞

范例：

const a=102,g=10;

const c=+0,d=+10,e=-0,f=0;//整数

const b=-23;

分析：无符号整数就是0到9或者以非0开头的数字串，“0152”不合法；

整数可以有符号，“+0”“-0”“+102”都是合法的；

＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

＜数字＞ ::= 0｜＜非零数字＞

范例：

int a1s；

char \_a1;

int \_\_b1;

int \_;

char \_1\_;

分析：标识符可以是单个字母，也可以是以字母开头后面字母数字结合的串

“int 5a；”非法，“int a5；”合法;

字母包含下划线，因此标识符可以用’\_’作为开头,单个’\_’也算标识符，范例中的几个标识符都符合文法；

4.

＜参数表＞ ::= ＜参数＞{,＜参数＞}| ＜空>

＜参数＞ ::= ＜类型标识符＞＜标识符＞

＜类型标识符＞ ::= int | char

范例：

int func(){...}

Void func1(int a,char c){...}

分析：函数的参数表可以为空，也可以包含多个参数，两个参数间用逗号隔开

5.

＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

＜语句列＞ ::= ｛＜语句＞｝

范例：

int func(参数表){

const int a=10; //常量说明

int c； //变量说明

/\*语句列

<语句>

<语句>

...

\*/

}

void func2(参数表){

<空>

}

...

分析1：复合语句，常量说明、变量说明可有可无，必须要有语句列；而且三者顺序不能改变；“char a;const int b=2；”不合法；

分析2：语句列可以包含0个或多个语句，语句可以为空，那这就说明函数的复合语句中常量说明、变量说明、语句列都可以没有，但如果出现这种情况这个函数就基本上没什么意义；

6.

＜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＞;

＜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＞

＜循环语句＞ ::= while '('＜条件＞')'＜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

＜情况语句＞ ::= switch '('＜表达式＞')' '{'＜情况表＞＜缺省＞ '}'

＜缺省＞ ::= default : ＜语句＞|＜空＞

＜情况表＞ ::= ＜情况子语句＞{＜情况子语句＞}

＜情况子语句＞ ::= case＜常量＞：＜语句＞

＜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＞

范例：

int sym,k；

int add(int a，int b){

return a+b；//返回语句

}

void add1(){

k=k+1;

return;//可有可无返回语句

}

void add2(){

k=k+2;

}

void main(){

const int a=2；

int c，d[5]，e[5]；

char s；

c=1；d[0]=1;e[0]=1;//赋值语句

scanf(sym); //读语句

while(c<5){

d[c]=d[c]+1;//赋值语句（使用表达式赋值）

if(c-1)k=k+1//条件是表达式

if(c==1)e[c]=d[c];//条件语句，将数组元素值赋给另外的数组元素

if(c>=2)

e[c]=add(e[c-1],e[c-2]);//赋值语句，调用有返回值函数语句

}/\*循环、条件嵌套\*/

swich(sym){ //swich情况语句

0: //情况子语句

add1();

1: //情况子语句

add2();//无返回值函数调用语句

default：//缺省

<语句>/<空>

}

; //空语句

printf（sym，k）；//写语句

}

分析1：赋值语句中，可以对变量标识符和数组元素赋值，可以将表达式的值赋给它们；

分析2：函数调用的值参数表是否为空或者有几个表达式都是由定义函数时定义了多少形参决定的，实参和形参的类型必须对应，个数也应相同，否则应报错例如:

“int k,b;

char e;

int func(int a,char c){复合语句略}

func(k,b,e);”这是不合法的；

分析3：语句可以为空：就如范例中的空语句；

分析4：作为条件的表达式，表达式为0时表明为假，否则为真；

分析5：情况语句表达式值可以是int或char，范例中是int型，情况表以及缺省都是不能缺失的；

由于没有’break’所以该文法设计每个情况子语句执行完后直接退出情况语句；

分析6：语句可以进行嵌套；

分析7：该文法对于条件语句，没有else，这样在使用if时要注意；

7.

＜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＞']'|'('＜表达式＞')'｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞

＜加法运算符＞ ::= +｜-

＜乘法运算符＞ ::= \*｜/

范例：

const int a1=2,b2=3;

const char cs=’s’,ss=’q’;

int a[2];

char b[2]；

int funct1(参数表略){函数主体部分略}

char funct2(参数表略){函数主体略}

a[0]=4,a[1]=5;

b[0]=’z’,b[1]=’A’;

/\*表达式\*/

+a1+b2-1

-a1-b2+1

-a1\*b2+2-a[1]\*a[0]+funct1

(a1+b2)\*2

(b2-a1)/2

cs+1+funct1+funct2

b[0]+a[0]+funct1+’a’

’a’\*b[0]

/\*表达式\*/

分析1：表达式可包含一个或多个项，第一个项之前有-表示是否需要将第一个项取负，+和没有符号效果相同；

后面的加法运算符用于决定加减运算；“1+-2”这种是不合法的；

分析2：当因子为数组元素时，’[’’]’之间可以是表达式，比如a[-1+’a’+a]是合法的（当然a这个数组应该要有足够多的元素，如果-1+’a’+a超出范围，则错误不合法）；

分析3：“i+k >= n+m”这种条件表达式不在该文法表达式的范围，因此不能将这种条件表达式的结果赋给其他变量：“b=(i+k >= n+m)”不合法；

分析4：根据表达式的文法说明字符也可以进行加减乘除运算，字符做运算时，转化为ASCII码运算，取整数值；

分析5：’/’表示除法，除法应取结果整数部分；

8.

＜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＞')'

范例：

const int a=1;

const char c=’s’;

int b；

char d；

scanf(b,d);

printf(”a+c is :”,a+c);

printf(”Hellow world!”);

printf(a+c);

分析1：scanf语句在对应位置与输入的值类型相对应，否则报错;const 类型变量的值应当无法改变；

分析2：写语句中，字符串原样输出，单个char类型的变量或常量输出字符，其他表达式按整型输出，这意味着’a’+1会输出98，’a’就输出a，整数就直接输出；

分析3：由于该文法没有换行定义，这就需要在设计的时候进行特别说明（为了方便应该设为自动换行）；

根据写语句文法，“printf(a,b,b)”这种是不合法的；

9.

＜返回语句＞ ::= return['('＜表达式＞')']

范例：

const int a=1;

int add(int s){

...略

return a+s;

}

int r1(){

...略

return 1;

}

void func(){

..略

}

void main(){

...略

return;

}

分析：有返回值函数return语句需要返回表达式；无返回值函数、主函数只用写return，或者省略。

10.

＜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＞ ::= \*｜/

＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

＜数字＞ ::= 0｜＜非零数字＞

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

范例：

const char a=’\_’,c=’s’//字符

” asd AS#$% a+-A ”//字符串

分析：字母包含下划线，const char a=’\_’是合法的；

字符串中也可以有空格；

11.

＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}｜＜空＞

范例：

int func1（int a，int b，char c）{

...复合语句

return 返回值

}

//调用

int a,b,e;

char ss ;

func1(a-ss,b+e\*a,ss+a);

分析：可以将表达式作为实参用于调用函数；

值参数表中，表达式个数、值类型必须和参数表中的形参定义相对应；