文法解读文档

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＞‘]’ )}

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

＜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＞‘}’

范例：

Const int a = 1,b=2; //常量说明部分

Const char c = ‘a’;

Int va,vb[4]; //变量说明部分

Char vc[3];

Int func1(int a, int b){ //有返回值函数定义

Int c;

C = a + b;

Return (c);

}

Void func2(){ //无返回值函数定义

Printf(“hello world!”);

}

Void main(){ //主函数

Int d = c –b;

Char e = ‘\*’;

Printf(e);

Func1(a,d);

Func2();

}

**分析1：**<程序>给定了我们C0文法程序的标准格式。我们不难看出程序的各部分声明是存在顺序的，不能随便改变声明的顺序。先声明变量后声明常量的情况是不被允许的，例如“int a;const int b”;此外，需要在主函数中调用的函数需要在主函数之前声明，不能再主函数之后继续做函数声明。

**分析2：**每一个程序按顺序由常量说明，变量说明，函数说明（有无返回值）和主函数组成，其中不同类型的声明不能交换顺序，但是同一种类的声明可以由多个。

**分析3：**变量说明，常量说明，有返回值函数定义以及无返回值函数定义并不是程序的必要组成部分，只有最后的主函数是必须存在的。

**分析4：**在进行常量说明的时候，不能再同一个const中声明两种不同的格式的常量。如果需要声明整型和字符型的常量，需要使用两个const来分别声明，如：“const int a, char b；”是错误的。但同类型的单个可以在同一个const中声明，如：“const int a, b;”是正确的。变量说明中也有相似的规律, 同一类型的单一变量和数组可以在一起进行声明.

**分析5:** 在函数声明的过程中, 有返回值和无返回值的函数声明并没有限定顺序, 也就是说二者可以进行交替声明, 但是必须在变量声明结束之后, 且在主函数声明之前.

**分析6:** 从常量说明和常量定义可以看出, 并没有常量数组.

1. ＜字符串＞ ::= "｛十进制编码为32,33,35-126的ASCII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＞ ::= \*｜/

＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==

＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

＜数字＞ ::= ０｜＜非零数字＞

＜非零数字＞ ::= １｜．．．｜９

范例:

Char a = ‘a’, b = ‘\*’, c = ‘+’, d = ‘0’ //字符

“This is a string!” //字符串

**分析1:** C0文法对字符的定义仅限于加法运算符, 乘法运算符, 字母和数字, 其他标点等符号均不算做字符, 关系运算符不算做字符. 字符声明时需要使用单引号.

**分析2:** 字符串可以包括字符所不包含的特殊符号, 声明时需要使用双引号包围. 字符串并不包括换行符.

**分析3:** 下划线符号’＿’算作字母.

1. ＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝

＜整数＞ ::= ［＋｜－］＜无符号整数＞｜０

范例:

15 //可以不带符号

+21 //可以带正号

-32 //可以带负号

**分析:** C0文法的整数不包括前导零, 符号并非强制. 整数0 前面不能加正负号.

1. ＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

范例:

a, b1,\_a0 //正确的标识符

1c //错误的标识符

**分析:** 标识符必须由字母开头, 后面可以添加数字, 但是数字开头是错误的.

1. ＜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＞‘]’ )}

范例:

Int a[233], b;

**分析1:** 同类型的变量声明可以在一起进行, 单一变量和数组变量的声明可以交叉进行, 但是需要注意填入数组中括号的只能是数字而不能是另一个变量.

**分析2:** 标识符区分大小写.

1. ＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞

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

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

＜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＞

＜类型标识符＞ ::= int | char

范例:

Int func1(int a, int b){ //有返回值函数定义, 带有参数

Int c;

C = a + b;

Return (c);

}

Void func2(){ //无返回值函数定义

Printf(“hello world!”);

}

**分析:** 进行函数声明的相关规则, 我们可以看出参数不一定必须带有参数, 参数只接受标识符, 并不接受数组类型的参数.

1. ＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

＜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＞;|＜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＞

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

＜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＞‘)’

＜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＞‘)’

＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}｜＜空＞

＜返回语句＞ ::= return[‘(’＜表达式＞‘)’]

范例:

Const int a = 1; //常量说明

Char c; //变量说明

Int d;

If(a > 0){ //条件语句

While(a > 0){ //循环语句

a = a – 1; //赋值语句

}

Scanf(c); //读语句

getchar(c); //有返回值的函数调用

voidfunc(); //无返回值的函数调用

d = c + 1;

switch(a){ //情况语句

case 1 : printf(“1”); //写语句, 情况子语句

case 0 : printf(“0”);

default: printf(“no”); //缺省

}

Printf(c);

Return (c) //返回语句

}

**分析1:** 上述复合语句为构成函数核心的主要部分, 可以看出常量和变量的声明需要在语句列之前完成, 其顺序不能颠倒.

**分析2:** 语句列可以由多种语句组成, 不同语句类型之间没有顺序关系. 需要注意的是, <空>的后面有一个分号, 这就意味着我们的编译器需要处理多余的分号.

**分析3:** 写语句中的字符串原样输出.

**分析4:** char类型的表达式，用字符的ASCII码对应的整数参加运算，在写语句中输出字符

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

＜情况表＞ ::= ＜情况子语句＞{＜情况子语句＞}

＜情况子语句＞ ::= case＜常量＞：＜语句＞

＜缺省＞ ::= default : ＜语句＞|＜空＞

＜常量＞ ::= ＜整数＞|＜字符＞

范例:

switch(a){ //情况语句

case 1 : { printf(“1”);

print(“2”);} //写语句, 情况子语句

case 0 : printf(“0”);

default: printf(“no”); //缺省

}

**分析1:** 情况语句中，switch后面的表达式和case后面的常量只允许出现int和char类型；每个情况子语句执行完毕后，不继续执行后面的情况子语句

**分析2:** C0的情况语句并没有break关键字, 这就意味着如果一个情况子语句下面对应了多条语句, 在读到下一个case/default之前的所有语句都同属于一个语句列, 在处理的时候需要注意.

1. ＜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

范例: a == 1 //单个表达式

1-1 //表达式之间做运算

**分析:** 条件语句的条件可以为单一表达式也可以为两个表达式之间运算得出的结果. 注意不同类型表达式进行计算时需要进行类型转换.

1. ＜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＞

范例:

+a //标识符当做表达式

-b[1] //数组标识符当做表达式

(c) //括号嵌套表达式

4 //正数和字符充当表达式

‘e’

Getsym(); //有返回值调用的函数

a + b[1] \* (c) //有项和因子之间运算所获得的复杂的表达式类型

分析: 这里规定了表达式的具体组成形式. 具体来说, 表达式由若干个项和加法运算符构成, 不同的项之间只允许有一个加法运算符. 第一个项之前的符号并不是必须的. 项有因子构成, 不同因子之前由乘法运算符相连接, 不同因子之间不允许出现两个乘法运算符. 因子可以由标识符, 标识符[表达式], (表达式), 整数, 字符, 有返回值的函数调用语句来构成. 这里需要注意的是, 如果一个标识符是数组类型, 那么他的下标可以是一个表达式, 这与变量声明时下标必须是一个无符号整数不同, 在编写编译程序的时候需要加以考虑.