# 文法解读

## 初始文法：

具体文法省略，每一条见下面分析。 附加说明： 1. 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2. 标识符区分大小写字母3. 写语句中的字符串原样输出4. 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5. 数组的下标从0开始6. for语句先执行一次循环体中的语句再进行循环变量是否越界的测试

### 程序&主函数

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＞

* 分析：说明程序的组成结构是，先常量说明(可出现一次或不出现)，然后是变量说明(可出现一次或不出现)，注意这个是先后顺序的，函数和过程的声明就没有，然后是可以出现0次或多次的有返回值或无返回值的函数的定义，最后是主函数。

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

* 分析：

正确范例：

void main() { return (); }

### 常量部分

1. ＜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＞}
2. ＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z ＜数字＞ ::= ０｜＜非零数字＞ ＜非零数字＞ ::= １｜．．．｜９ ＜字符＞ ::= ‘＜加法运算符＞’｜‘＜乘法运算符＞’｜‘＜字母＞’｜‘＜数字＞’

* 分析：常量说明部分至少有一个const。然后每个常量定义都是意义配对的关系，即**const后面的标识符的类型要和后面的常量值对应。**
* 此外，注意到字符是有限制的，只允许所有的大小写字母和\_，然后字符只有字母和数字和+-\*/。
* 正确范例：
* const int N = 100, M = -10000; const char ch = 'H'; const int K = 0; const char ch1 = '+', ch2 = '-', ch3 = '\*', ch4 = '、', ch5 = '\_', ch6 = 'a', ch7 = '0', ch8 = '"', ch9 = '9';
* 错误范例：
* const char cc = 'cc'; const int NN = 'C'; const int MM = 2.0; const int KK = KK, chhc = '2';

### 变量部分

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＞‘]’ } ＜类型标识符＞ ::= int | char

* 分析：变量说明部分至少要有一个变量定义。在变量定义里，至少有一项是一个定义。而一个定义是由一个标识符或者标识符加上数组下标组成的。
* 以及，好像我们这个文法有点问题，其它组的有一个是这样的＜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＞']' )}，就是最后他们那个逗号后面的内容少了一个括号括起来，或者说是|后面的也少个逗号。这样的问题就是int ab[10]，在我们的文法里可以a分配给第二项，b[10]给第三项，然而这个肯定不是我们想看到的。
* 正确范例：
* int a[0], b[100], c; char str[10], chr, s[10000];
* 错误范例：
* int n = 1; char s[n];

### 语句&语句列&复合语句

1. ＜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＞;
2. ＜语句列＞ ::= ｛＜语句＞｝
3. ＜复合语句＞ ::= ［＜常量说明＞］［＜*变量*说明＞］＜语句列＞

分析：语句可以是条件语句或者循环语句或者赋值语句。需要注意的是，由于赋值语句、读语句、写语句返回语句、调用语句后面没有分号，因此我们需要在后面加上分号。然后语句列是0个或多个语句组成的，然后语句列需要用{ }来包住。**复合语句如果同时有常量说明和变量声明，那么常量说明在变量说明的前面。**

正确范例：

const int i = 1;  
const char h = '2';  
int n,m;

错误范例：

int n,m;  
const char h = '2';

#### 赋值语句

1. ＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞

* 分析：赋值语句的形式是标识符 = 表达式。
* 正确范例：
* x = x; a[x] = c; a[0] = 100;

#### 条件语句

1. ＜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

* 分析：条件是由表达式或者一个关系运算符连接两个表达式构成的，然后条件语句必须有条件，可以没有else。
* 正确范例：
* ``` if (a=100);
* if (ch > ‘a’); else; ```

#### 循环语句

＜循环语句＞ ::= for‘(’＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝＜标识符＞(+|-)＜步长＞‘)’＜语句＞＜步长＞ ::= ＜非零数字＞｛＜数字＞｝

分析：这个for循环与一般C语言的语法不同的有：首先不能有for (int i…这种写法，然后不能有i++，i=j,i=j+0这种语法。以及，**条件不能为空**。

正确范例：

for (i = 1; i <= 100; i = i+1)  
 ans = 0;

错误范例：

for (i = 1; i <= 100; i = i-0);  
for (int i = 1; i <= 100; i = i+1);

### 表达式

＜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＞‘}’

分析： 因子是我们这里面最小的单元，有五种，标识符、数组索引、整数/字符、函数调用语句、(表达式)组成。 项是由因子后接上若干个\*/因子组成。即可以在因子上套很多个乘除因子。 表达式是可以在最前面加上正号或负号然后一个项再若干个+-项组成的。即最前面可以有+或-，然后套很多个项。

正确范例：

+-1  
--1  
+123\*+123/-456\*a[10]\*'a'/mymax(1,2)\*(-1-2)

### 标识符&整数

＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝＜整数＞ ::= ［＋｜－］＜无符号整数＞｜０＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

分析：无符号整数相当于就是正整数且没有正号，并且没有前导0。然后整数的概念就相当于有正负号的无符号整数或者0（**因此整数不允许有+0，-0，前导0**）

### 函数（包括有返回值和无返回值）

1. ＜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＞‘}’
2. ＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞3. ＜参数＞ ::= ＜参数表＞＜参数表＞ ::= ＜类型标识符＞＜标识符＞{,＜类型标识符＞

分析：无返回值的函数相当于C语言里的过程，void. **然后这里我们需要考虑的一点是：如果有返回值的函数没有return 一个返回值或者return ;该怎么处理。我看去年的帖子是说语法分析无需检查，但语义分析需要处理。**因此之后要注意一下这个问题。

以及： 根据老师讨论区的回复，**不允许出现void work()或者int mymax()，即无参数必须是空，调用的时候也是**

正确范例：

int find(char c1,char c2) {return (0);}  
int mymax { int x = 1; return (x); }  
void work {}  
void work\_2 {return ;}

错误范例1:（某一个分支没有返回值，坑!）

char job{  
 if (n > 10){  
 return 'Y';  
 }  
}

错误范例2: (void不允许有返回值)

void job2{  
 if (n > 10){  
 return 'Y';  
 }  
}

错误范例3:（有返回值函数必须得返回一个真值）

int job3{  
 return ;  
}

### 情况语句

＜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＞

注意到switch后面的表达式和case后面的常量只允许出现int和char类型，而且如果有一条规则匹配了，就不需要继续匹配了。这里需要注意，如果情况子语句是空的情况。

### 函数调用语句

＜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＞

分析：和C语言非常类似。

正确范例：

work()  
mymax(1,2)  
find(str, "123");  
}

### 读写语句&返回语句

1. ＜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＞‘)’
2. ＜返回语句＞ ::= return[‘(’＜表达式＞‘)’]
3. ＜字符串＞ ::= “｛十进制编码为32,33,35-126的ASCII字符｝”

分析：读语句不需要参数列表，然后可以一次读入多个变量。输出语句允许出现字符串和表达式的组合体。返回语句有return和0个或1个表达式组成。**以及需要强调的是，输出语句的字符串有很大限制，一方面是没有回车(???)，然后ASCII为32的是空格，33为感叹后，去掉的34是双引号，很大程度上减少了我们处理转义字符的麻烦，但单引号(ascii码值为39)的仍需考虑。**

正确范例：

scanf(n, str)  
printf("str = ", str)  
return (str)  
return ()