编译文法解读

14061024 刘恒睿

1. ＜加法运算符＞ ::= + | -

   ＜乘法运算符＞ ::= * | /

   ＜关系运算符＞ ::= < | <= | > | >= | != | ==

   分析：

   这三条文法指明了程序里面的基本运算符都有哪些。

   示例：

```
a = b + c;
a = b - c;
a = b * c;
a = b / c;
d = a > c;
d = a < c;
...
```

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＞'

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

   分析：

   这5条文法说明了程序中的数字和字符是如何定义的，也就是标识符可以由下划线加字母数字组成，具体语法规则见后。特别说明的是字符需要单引号而字符串是双引号。而字符串中的

   示例：

```
字母：_ A B C D ... x y z
数字：0 1 2 3 4 5 6 7 8 9
字符：'A' 'B' ... '+' '-' '*'
字符串："124dsfdgg !_#(){{"
```

3. ＜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：

   该文法定义了程序，他可以用一个常量说明，一个变量说明，任意多个函数定义以及必须有一个主函数

   示例：

```
const int a;
int b;
void c();
int d();
void main(){}
```

4. ＜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＞}

分析：

这两条语法说明了常量部分的定义，常量可以定义一到多个，定义中首先包括了一个常量说明符`const`，然后是`int`或者`void`两种数据类型，分别对应上面的整数和字符两种语法。

示例：

```
const int a = 10; const char b = 'digit', c = 'digit1'; const int d = 100 , f = 1000;
```

5. ＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝

   ＜整数＞ ::= ［＋｜－］＜无符号整数＞｜ 0

分析：

这两句语法定义了整数。

示例：

```
+100 -123 34 0 234012300
```

6. ＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

分析：

语法定义了标识符由数字和字母组成，数字不能作为开头。

示例：

```
_A1234 sfb_d12
```

7. ＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞

分析：

说明了如何声明头部。

示例：

```
int a
char b
```

8. ＜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＞‘]’ }

> ＜常量＞ ::= ＜整数＞|＜字符＞
>
> ＜类型标识符＞ ::= int | char

分析：

以上语法说明变量和常量的定义。变量定义可以有一到多个，定义首先要说明类型标识符，然后是变量名（标识符），变量可以定义数组，数组大小为一个无符号整数。

示例：

```
int a;char b[10];int c,d,e[1101]
```

9. > ＜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＞‘}’
   >
   > ＜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：

这5条语法定义了函数的声明方式，分为有返回值和无返回值的函数声明。区别在于是`void`还是`int|char`。函数还包括一个参数定义以及复合语句块定义。参数允许没有。

示例：

```
void a(int a,char b)
  {
    ...
  }
int b(char c)
  {
    char d;
    ...
    return d;
  }
char c(){}
```

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

分析：主函数为固定的定义方式。

示例：

```
void main()
{
  ...
}
```

11. > ＜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：

这里定义表达式的语法。表达式开头可出现`+/-`号，计算过程中可以有标识符之间的计算，可以计算数组中的值，可以带入函数返回值进行计算，也可以计算整数或者字符。

示例：

```
++1-+2*c-(d(e,f,g+h))/16*101+'a'-'s'*i[12]
```

12.

> <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>［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
>
> <循环语句> ::= while ‘(’<条件>‘)’<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>
>
> <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>
>
> <语句列> ::= ｛<语句>｝
>
> <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>‘)’
>
> <返回语句> ::= return[‘(’<表达式>‘)’]

分析：

以上16条语法定义了语句。语句可分为条件语句，循环语句，语句列，函数调用语句，赋值语句，读写语句，情况语句，返回语句。

- 条件语句

条件语句由`if`开头，然后必定存在`(`<条件>`)`，条件中可以只含一个表达式或者两个表达式做关系运算。表达式为0条件为假，否则为真。之后可以含有一个`else`语句块，也可以没有。

示例：

```
if (a > b)
  c = a;
else
  c = b;
/****************/
if (a)
  {
    c = a;
  }
/****************/
if(++1-+2*c-(d(e,f,g+h))/16*101+'a'-'s'*i[12] <= 12+'A')
  {
    b=a;
    c=b;
  }
else
  {
    d = c;
    e = d;
  }
```

- 循环语句

  循环语句以`while`开头，包含一个`(`<条件>`)`，条件描述与`if`语句没有区别。

  示例：

  ```
    while(c)
      {
        c = c - 1;
      }
    /****************/
    while(c+d>d(e,f,g))
      d = c;
  ```

  - 语句列

    语句列为由`{}`包裹起来的一组语句。

    示例：

    ```
    {
      a = c;
      b = 'b';
      c = c+a*b/f;
    }
    ```

  - 函数调用语句

    函数调用语句为调用一个函数所需要的语句。

    示例：

```
d(a,s,d,f)
a()
```

- 赋值语句

  赋值语句为用一个表达式给一个标识符赋值，可以给数组中的一个元素赋值。

  示例：

```
a = 100;
b[1] = ++1-+2*c-(d(e,f,g+h))/16*101+'a'-'s'*i[12];
```

- 读写语句

  读语句以`scanf`开头，可以用来读取1个或多个值到标识符中，不能直接将值读入到数组中的某一个中。

  写语句以`printf`开头，可以打印字符串和表达式，但是不能打印单个字符。

  示例：

```
scanf(a,b,c);
scanf(a);
printf("asdf!_ #",++1-+2*c-(d(e,f,g+h))/16*101+'a'-'s'*i[12]);
printf("asd");
printf(++a-+b*c);
```

- 情况语句

  情况语句由`switch`开头，包含一个`(`<条件>`)`，然后由`{}`包裹一个情况表和一个可有可无的缺省。情况表中可以含有多个情况子语句。情况子语句由`case`开头，不存在break语句。

  示例：

```
switch (c+d>d(e,f,g))
  {
    case 1:
        a = b + c;
    case 'a':
        {
          a = b + c;
          c = d;
          e = foo(a,x,c);
        }
    default:
        a = 0;
  }
/*********************/
switch (c+d>f)
  {
    case 1:
        a = b + c;
  }
```

- 返回语句

用作函数返回值，以`return`开头。

示例：

```
return(a);
/**************/
return(b + c);
/**************/
return(d(e,f,g));
```