**C类文法解读作业**

周美廷-76066002

首先文法说明中，{} 代表多于等于0， 而 [] 代表可有可无。

**<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>**

C类文法的基层就是程序本身。这个程序是由多个成员组成的:

1. 常量说明（可有可无）
2. 变量说明（可有可无）
3. 有返回值函数定义**亦或者**无返回值函数定义（可有可无）
4. 主函数（必须）

一个程序中会常常用到的有：

1. 字母

**<字母> ::= \_|a|...|z|A|...|Z**

1. 数字

**<数字> ::= 0|<非零数字>**

**<非零数字> ::= 1|...|9**

1. 字符

以 ’ 开头，跟着可以是字母或者数字，然后后面要有 ’

**<字符> ::= '<加法运算符>'|'<乘法运算符>'|'<字母>'| '<数字>'**

1. 字符串

以 ” 开头，跟着可以十进制编码中为32，33，35-126中的任何ASCII字符组成。

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

# **数据类型**

C类中有这几种数据类型：

1. 常量（const）
2. 变量（var）
3. 函数（function）

* 有返回值函数定义
* 无返回值函数定义
* 主函数

变量与常量都具有公用的数据类型文法：

1. 标识符

每个数据类型定义都有标识符，标识符必须是小于等于11个字符组成，必须以字符开头，可以有大写或小写字母也可以有数字。

**<标识符> ::= <字母>{<字母>|<数字>}**

1. 整数

整数（integer）可以是负整数，正整数，或是0

**<整数> ::= [+|-]<无符号整数>**

**<无符号整数> ::= <非零数字>{<数字>}|0**

## **常量说明部分**

条件/规定：

1. 它们在每一个程序中必须定义在变量说明部分
2. 常量是不可变的，也就是说，定义后不能再赋值
3. 可以同时定义多于一个常量，以逗号分
4. 文法：

**<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>}**

例子： const int a = 1;

const int b = 3, c = 5;

const char d = ‘e’;

## **变量说明部分**

首先，定义变量时需要定义它的类型（int或char）。然后变量可以是单个或者array。若是单个变量，只需写下该变量的标识符。若是array类型，需要在标识符后面加上**[<无符号整数>]**。

条件/规定：

1. 它们在每一个程序中必须定义在常量说明部分
2. 可以同时定义多于一个变量，以逗号分
3. 文法：

**<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>‘]’)}**

**//<无符号整数>表示数组元素的个数，其值需大于0**

**<类型标识符> ::= int|char**

例子： int a;

int array[10];

int b, c, f[6];

char d;

char m[7];

## **函数**

C类文法中的函数定义分为三大种：

有返回值函数定义，无返回值函数定义，以及主函数。

一个程序中，有返回值和无返回值函数定义是可有可无的，而主函数是必须。

函数定义中的参数表是用于传参数，它可以是空的也可以有多个参数。

**<参数表> ::= <类型标识符><标识符>{,<类型标识符><标识符>} | <空>**

每个函数定义会有一系列计算语句—复合语句。复合语句中可以再次定义常量和变量，当然这些常量和变量都是局部性的。

**<复合语句> ::= [<常量说明>] [<变量说明>] <语句列>**

1. 有返回值函数定义

有返回值函数会返回一个值，int或char，所以该函数的类型也必须是int或者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>**

1. 无返回值函数定义

无返回值函数不会返回任何值而只是用于计算，该函数的类型是void。

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

1. 主函数

主函数就是程序最初执行的函数，不论主函数写在其他函数定义的前面或后面，每个程序会一直从主函数开始。

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

# **表达式**

## **语句**

语句可分为几种语句，分别是赋值语句，条件语句，repeat循环语句，for循环语句，过程调用语句，复合语句，读语句，写语句。

**<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>;**

**<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**

### **条件语句**

**<条件语句> ::= if ‘(’<条件>‘)’<语句> [else <语句>]**

条件语句允许我们执行某个（些）代码如果符合我们给的条件。若不符合，那就执行其他代码段。条件语句中一般会用关系运算符来判断条件：

**<关系运算符> ::= <|<=|>|>=|!=|==**

* 1. < (小于)
  2. <= (小于等于)
  3. > (大于)
  4. >= (大于等于)
  5. != (不等于)
  6. == (等于)

例子： if(x<1) x=x+2;

if(0==1) {printf(“error!”);}

if(x!=y) x=y; else x=1;

### **循环语句**

C类文法中的循环语句分为3种，While，do-While，For。

While与Do-While一样是比较简单的条件循环语句。两者的区别就在与：

* Do-While循环会先执行一次语句在检查条件，若符合就会再次执行该语句。
* While循环会先检查条件，若符合才会执行语句

For语句中我们可以更好的控制该循环语句的性质，我们可以先初始化某个（些）变量，然后检查条件，后还可以控制循环语句的步长。

1. While循环

**<循环语句> ::= while‘(’<条件>‘)’<语句>**

1. Do-While循环

**<循环语句> ::= do <语句>while ‘(’<条件>‘)’**

1. For循环

**<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> ::= <无符号整数>**

例子： while(x<10) x=x+1;

do x=x+1; while(x<9)

for(i=0; i<10; i=i+1)

### **函数调用语句**

在一个函数中，我们可以调用其他函数执行那函数中的算法语句。调用时，我们可以传参数（写在参数表里）。

**<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>**

例子： addFunc(1, 2);

runThis();

### **赋值语句**

**<赋值语句> ::= <标识符> = <表达式> | <标识符>‘[’<表达式>‘]’= <表达式>**

正如它的名字，赋值语句的作用是用来给变量赋值的（表达式）。

**<表达式> ::= [+|-]<项>{<加法运算符><项>} // [+|-]只作用于第一个<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>**

赋值过程一般会用以下运算符来进行赋值：

1. 加法运算

**<加法运算符> ::= +|-**

1. 乘法运算

**<乘法运算符> ::= \*|/**

例子： x = x+1;

### **读语句**

读语句是用来从某个设备（键盘，文件，scanner等）中读取它的内容。语法为：

**<读语句> ::= scanf'('<标识符>{,<标识符>}')'**

例子：scanf(d);

### **写语句**

写语句是用来把某些内容写到某个设备（显示器，文件等）。语法为：

**<写语句> ::= printf '(' <字符串>,<表达式>')'| printf '(' <字符串>')'| printf'('<表达式>')'**

例子：printf(“Enter the radius of the circle”);

### **返回语句**

返回语句是用于结束某个函数的执行。‘返回’就是代返回该函数执行的前一个状态。

**<返回语句> ::= return[‘(’<表达式>‘)’]**

例子： return 0;

return (1+2);