Min-C语言语法基本规则需求说明

（2017年8月28日）

说明：下列定义为示意性说明，不是严格的语法定义。

* 文法描述符号

=：（定义为），|（或），+（连接,可省），\*（0次或多次重复），

(，)，<，>（语法成份），[，]（可选项）

* min-C语言基本语法成份

**基本字符集：**

<字母> =：a | b | … | z | A | B | … | Z

<数字> =：0 | 1 | … | 9

<小数点> =：.

<符号> =：= | + | - | \* | / | $ | ( | ) | { | } | ” | > |< | , | ; | \ | 空格 | [TAB]

<标识符> =：<字母> [ <字母>|<数字> )]\*

含义：标识符是首字符为字母的、由字母或数字组成的符号串。

<保留字> =：int | real | string | while | if | else | out | in

void | return

保留字是系统已定义的具有特定含义的标识符，不能重新定义含义。

<整型常量> =：[ + | - ] <数字><数字>\*

<实型常量> =：[ + | - ] <数字><数字>\* <小数点>[<数字>]\*

<单字符> =：<字母> | <数字> | <小数点> | <符号> | <…>

<转义符> =：\n | \0 | \\ | \” | \<字母>

<串> =：”[ (<单字符>|<转义字符>)\* ]”

<数据类型> =：int | real | string

**数据类型：2**字节整型，4字节实型，串类型string

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 字节数 | 在解释器中对应的数据类型 | 数值范围 |
| int | 2 | c++中的short | -32768～+32767 |
| real | 4 | c++中的float，小数点后精确的位数为(6-小数点前的位数), 小数点后的位数不足(6-小数点前的位数)时不补零 | -3.4\*10^38～+3.4\*10^38 |
| string | 不限制 | c++中的string | 无 |

注：string 不定义长度 理论上不超出内存限制的string长度都是合法的

如果用户定义 real r=1.123456789123;

则r在解释器内部实际上为1.1234567

<变量名> =：<标识符>

**变量名：**长度不超过8的标识符（字母大小写敏感）。

**变量：**应先定义后使用。不定义直接使用默认为int变量

不支持在定义时初始化。

支持相同数据类型的变量在定义时采用逗号分隔的形式：<变量类型> <变量名>[,<变量名>]\* ;

<函数名> =：<标识符>

**函数名：**长度不超过8的标识符（字母大小写敏感）。

**支持的基本运算：**

单目运算：+正 | -负

算术运算：=赋值 | +加 | -减 | \*乘 | /除 | %余

串操作：$连接 | #删一个字符

minC语言中的串在解释器中的存储形式为c++中的string类

$支持整形和字符串之间的连接以及实型和字符串之间的连接

例1：s=79$”minC”，结果为s=”79minC”

例2：s=7.9$”minC”，结果为s=”7.9minC”而不是7.90000minC

例3：s=7.123456789$”minC”，结果为s=”7.12345minC”。解释器内部用cout输出浮点数总长为7位。

例4：s=123.123456$”minC”，结果为是s=”123.123minC”。

#含义是从原串中删除所有子串，若子串不存在则返回原串

例1：s=”minCpyminCpyminC”#”py”，结果为s=”minCminCminC”

例2：s=”minCpyminCpyminC”#”yp”，结果为s=”minCpyminCpyminC”

注意：$和#这两种串操作均是在原串的副本上进行的操作，操作结果不影响原串的值

例如 s1=”minC”; s2=”min”; s3=s1$s2; s4=s1#s2;

运行结果为：s1=”minC”,s2=”min”,s3=”minCmin”,s4=”C”;

解释器内部经过重载，使得串相关的操作只有$ | #。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 优先级 | 运算符 | 名称或含义 | 使用形式 | 结合性 | 说明 |
| 1 | () | 圆括号 | （表达式）/函数名(形参表) | 左到右 |  |
| 2 | - | 负号运算符 | -表达式 | 右到左 | -的二义性处理：如果符号-左侧没有操作数也没有操作符或者-左侧的操作符与-不同，则认为符号-为负号运算符，否则为减号如3--1(语法错误)-3+1(-:负号,+:加号)3+-1(+:加号,-:负号) |
| + | 正号运算符 | +表达式 | +的二义性处理：同上 |
| ! | 逻辑非运算符 | !表达式 | 单目运算符 |
| 3 | / | 除 | 表达式/表达式 | 左到右 | 双目运算符 |
| \* | 乘 | 表达式\*表达式 | 双目运算符 |
| % | 余数（取模） | 整型表达式/整型表达式 | 双目运算符 |
| 4 | + | 加 | 表达式+表达式 | 左到右 | 双目运算符 |
| - | 减 | 表达式-表达式 | 双目运算符 |
| 5 | $ | 串连接 | 串1 $ 串2 | 左到右 | 双目运算符 |
| # | 子串删除 | 串#子串 | 双目运算符 |
| 6 | > | 大于 | 表达式>表达式 | 左到右 | 双目运算符 |
| >= | 大于等于 | 表达式>=表达式 | 双目运算符 |
| < | 小于 | 表达式<表达式 | 双目运算符 |
| <= | 小于等于 | 表达式<=表达式 | 双目运算符 |
| 7 | == | 等于 | 表达式==表达式 | 左到右 | 双目运算符 |
| <> | 不等于 | 表达式!= 表达式 | 双目运算符 |
| 8 | && | 逻辑与 | 表达式&&表达式 | 左到右 | 双目运算符 |
| 9 | || | 逻辑或 | 表达式||表达式 | 左到右 | 双目运算符 |
| 10 | = | 赋值运算符 | 变量=表达式 | 右到左 |  |
| 11 | , | 逗号运算符 | 变量,变量,… | 左到右 | 不支持用逗号分隔表达式 |

关系运算：>大于 | <小于 | >=大于等于 | <=小于等于 | ==恒等 | <>不等

逻辑运算：&&与 | ||或 | !非

关系和逻辑运算结果（条件）：真为int型1，假为int型0；int型非0为真，int型0为假。

**语句：**分为简单语句和复合语句，简单语句由分号(;)结束。

**简单语句：**组成程序的基本单位。

**复合语句：**以 { } 包围起来的多个简单语句或复合语句，在语法上等价于一个简单语句。

**注释语句：**行注释，以 // 开头的字符串。

程序基本结构：函数

**<程序>=：<主函数> [<函数>]\***

**<主函数>=：**

int main( )

{ [ <语句> ; ]\*

}

**<函数>=：**

<数据类型> <函数名>( <参数列表> )

{ [ <语句> ; ]\*

}

函数体内必须有return语句表示函数调用结束

函数没有返回值时，函数返回值类型为void，函数体内用return ;表示调用结束。

**<语句>**=：<说明语句> | <可执行语句>

**<说明语句>**=：<数据类型> <变量名> [ , <变量名> ]\*

**<可执行语句>**=：<赋值> | <循环> | <判断> | <输入> | <输出> | <调用>

**<参数列表>** =：空格 | <数据类型> <变量名> [ , <数据类型> <变量名> ]\*

**<赋值>** =：<变量> = <算术表达式>

例如：a = b = 128;

c = a+b;

**<输入> =**：in [”<提示串>”,] <变量名>;

例1：

in a;

运行动作：从键盘输入一个整数存入变量a中

例2：

in ”请输入高度: ”, h;

运行动作：显示提示信息，等待键盘输入一个整数存入变量h中

**<输出>=**：out ([”提示串” ,] <变量名>) | ([<变量>|<整型常量>,] ”提示串”);

例1：

out a;

运行动作：显示变量a的值。

例2：

out ”Start\n”;

运行动作：显示1遍提示信息。

例3：

out 10, ”Start\n”;

运行动作：显示10遍提示信息。

例4：

out n, ”\*”;

运行动作：显示n个星号(\*)。

例5：

out ”a = ”, a;

运行动作：显示提示信息和变量a的值。

**<判断>**=：if <条件表达式> { 语句 }

[ else { <语句> } ]

限制：判断嵌套不超过5层。

例1：

if a==b

{

out ”变量a等于变量b，a=”, a; // 输出提示信息和变量a的值

out ”\n”;

}

例2：

if a>=b

{ out ”变量a大于等于变量b.\n”; // 输出提示信息

}

else

{ out ”变量a小于变量b.\n”;

}

例3：

if a>b {

out ”变量a大于变量b.\n”;

}

else {

if a==b {

out ”变量a等于变量b.\n”;

}

else {

out ”变量a小于变量b.\n”;

}

}

**<循环>**=：while <条件表达式> { <语句> }

限制：循环嵌套不超过5层。

例1：

k = 0;

while k<10

{

sum = sum+k;

k = k + 1;

}

if /while内形成新的作用域。if/while内新定义的变量会覆盖该if/while所在函数内的同名变量，作用域为该if/while语句。

**<调用>=：**

<函数名>( 实际参数列表 );

函数支持重载和递归

1. 函数间参数如何传递？

函数间采用值传递的方式进行参数传递。

1. 函数间如何进行变量保护，变量作用域？

解释器不支持全局变量，在调用函数时会形成新的变量作用域，在函数内部只能检测到该函数中新定义的变量。函数调用结束时，这一新形成的变量作用域随即消失。

1. 实际参数是否可以是表达式？

解释器支持表达式作为实际参数。

1. 返回值如何处理？

对于返回值非空的函数，可以把调用函数时得到的返回值用于调用方内的赋值语句，也可以作为一个单独的表达式而不被任何变量接收，也可以作为另一个函数的参数，但不可以作为out语句的输出。