

# C-- (Simplified) 语言文法

在本附录中，我们给出 C-- (Simplified) 语言的文法定义和补充说明。

## A.1 文法定义

### A.1.1 Tokens

INT  $\rightarrow$  /\* A sequence of digits without spaces <sup>①</sup> \*/

FLOAT  $\rightarrow$  /\* A real number consisting of digits and one decimal point. The decimal point must be surrounded by at least one digit <sup>②</sup> \*/

ID  $\rightarrow$  /\* A character string consisting of 52 upper- or lower-case alphabetic, 10 numeric and one underscore characters. Besides, an identifier must not start with a digit <sup>③</sup> \*/

SEMI  $\rightarrow$  ;

COMMA  $\rightarrow$  ,

ASSIGNOP  $\rightarrow$  =

RELOP  $\rightarrow$  > | < | >= | <= | == | !=

PLUS  $\rightarrow$  +

MINUS  $\rightarrow$  -

STAR  $\rightarrow$  \*

DIV  $\rightarrow$  /

AND  $\rightarrow$  &&

OR  $\rightarrow$  ||

DOT  $\rightarrow$  .

NOT  $\rightarrow$  !

TYPE  $\rightarrow$  int | float

LP  $\rightarrow$  (

---

①你需要自行考虑如何用正则表达式表示整型数，可以假设每个整型数不超过 32 位。

②你需要自行考虑如何用正则表达式表示浮点数，可以只考虑符合 C 语言规范的浮点常数（参见补充说明）。

③你需要自行考虑如何用正则表达式表示标识符，可以假设每个标识符长度不超过 32 个字符。

RP  $\rightarrow$  )

$LB \rightarrow [$   
 $RB \rightarrow ]$   
 $LC \rightarrow \{$   
 $RC \rightarrow \}$   
 $STRUCT \rightarrow \text{struct}$   
 $RETURN \rightarrow \text{return}$   
 $IF \rightarrow \text{if}$   
 $ELSE \rightarrow \text{else}$   
 $WHILE \rightarrow \text{while}$

用于 LL(1)实验的简化版 C—语法

### A. 1. 2 High-level Definitions

$\text{Program} \rightarrow \text{ExtDefList}$   
 $\text{ExtDefList} \rightarrow \text{ExtDef ExtDefList}$   
 $\quad | \ \epsilon$   
 $\text{ExtDef} \rightarrow \text{Specifier FunDec Compst}$

### A. 1. 3 Specifiers

$\text{Specifier} \rightarrow \text{TYPE}$

### A. 1. 4 Declarators

$\text{VarDec} \rightarrow \text{ID}$   
 $\text{FunDec} \rightarrow \text{ID LP RP}$

### A. 1. 5 Statements

$\text{CompSt} \rightarrow \text{LC DefList StmtList RC}$   
 $\text{StmtList} \rightarrow \text{Stmt StmtList}$   
 $\quad | \ \epsilon$   
 $\text{Stmt} \rightarrow \text{Exp SEMI}$   
 $\quad | \ \text{RETURN Exp SEMI}$

### A. 1. 6 Local Definitions

$\text{DefList} \rightarrow \text{Def DefList}$   
 $\quad | \ \epsilon$   
 $\text{Def} \rightarrow \text{Specifier Declist SEMI}$

DecList  $\rightarrow$  Dec

Dec  $\rightarrow$  VarDec ASSIGNOP Exp

### A. 1.7 Expressions

Exp  $\rightarrow$  LP Exp RP

| NOT Exp

| INT

| ID

## A. 2 补充说明

### A. 2.1 Tokens

这一部分的产生式主要与词法有关：

1) 词法单元 INT 表示的是所有（无符号）整型常数。一个十进制整数由 0~9 十个数字组成，数字与数字中间没有如空格之类的分隔符。除“0”之外，十进制整数的首位数字不为 0。例如，下面几个串都表示十进制整数：0、234、10 000。为方便起见，你可以假设（或者只接受）输入的整数都在 32 位之内。

2) 整型常数还可以以八进制或十六进制的形式出现。八进制整数由 0~7 八个数字组成并以数字 0 开头，十六进制整数由 0~9、A~F（或 a~f）十六个数字组成并以 0x 或者 0X 开头。例如，0237（表示十进制的 159）、0xFF32（表示十进制的 65330）。

3) 词法单元 FLOAT 表示的是所有（无符号）浮点型常数。一个浮点数由一串数字与一个小数点组成，小数点的前后必须有数字出现。例如，下面几个串都是浮点数：0.7、12.43、9.00。为方便起见，你可以假设（或者只接受）输入的浮点数都符合 IEEE754 单精度标准（即都可以转换成 C 语言中的 float 类型）。

4) 浮点型常数还可以以指数形式（即科学记数法）表示。指数形式的浮点数必须包括基数、指数符号和指数三个部分，且三部分依次出现。基数部分由一串数字（0~9）和一个小数点组成，小数点可以出现在数字串的任何位置；指数符号为“E”或“e”；指数部分由可带“+”或“-”（也可不带）的一串数字（0~9）组成，“+”或“-”（如果有）必须出现在数字串之前。例如 01.23E12（表示  $1.23 \times 10^{12}$ ）、43.0e-4（表示  $43.0 \times 10^{-4}$ ）、.5E03（表示  $0.5 \times 10^3$ ）。

5) 词法单元 ID 表示的是除去保留字以外的所有标识符。标识符可以由大小写字母、数字以及下划线组成,但必须以字母或者下划线开头。为方便起见,你可以假设(或者只接受)标识符的长度小于 32 个字符。

6) 除了 INT、FLOAT 和 ID 这三个词法单元以外,其他产生式中箭头右边都表示具体的字符串。例如,产生式  $TYPE \rightarrow int \mid float$  表示:输入文件中的字符串“int”和“float”都将被识别为词法单元 TYPE。

## A.2.2 High-level Definitions

这一部分的产生式包含了 C——语言中所有的高层(全局变量以及函数定义)语法:

1) 语法单元 Program 是初始语法单元,表示整个程序。

2) 每个 Program 可以产生一个 ExtDefList,这里的 ExtDefList 表示零个或多个 ExtDef(像这种 xxList 表示零个或多个 xx 的定义下面还有不少,要习惯这种定义风格)。

3) 每个 ExtDef 表示一个全局变量、结构体或函数的定义。其中:

a) 产生式  $ExtDef \rightarrow Specifier \ ExtDecList \ SEMI$  表示全局变量的定义,例如 `int global1, global2`。其中 Specifier 表示类型,ExtDecList 表示零个或多个对一个变量 的定义 VarDec。

b) 产生式  $ExtDef \rightarrow Specifier \ SEMI$  专门为结构体的定义而准备,例如 `struct{ . . . }`。这条产生式也会允许出现像“`int;`”这样没有意义的语句,但实际上在标准 C 语言 中这样的语句也是合法的。所以这种情况不作为错误的语法(即不需要报错)。

c) 产生式  $ExtDef \rightarrow Specifier \ FunDec \ CompSt$  表示函数的定义,其中 Specifier 是返回类型,FunDec 是函数头,CompSt 表示函数体。

### A. 2. 3 Specifiers

这一部分的产生式主要与变量的类型有关：

1) Specifier 是类型描述符，它有两种取值，一种是 Specifier→TYPE，直接变成基本类型 int 或 float，另一种是 Specifier→StructSpecifier，变成结构体类型。

2) 对于结构体类型来说：

a) 产生式 StructSpecifier→STRUCT OptTag LC DefList RC：这是定义结构体的基本格式，例如 struct Complex{int real,image;}。其中 OptTag 可有可无，因此也可以这样写：struct {int real,image;}。

b) 产生式 StructSpecifier→STRUCT Tag：如果之前已经定义过某个结构体，比如 struct Complex (...)，那么之后可以直接使用该结构体来定义变量，例如 struct Complex a,b;，而不需要重新定义这个结构体。

### A. 2. 4 Declarators

这一部分的产生式主要与变量和函数的定义有关：

1) VarDec 表示对一个变量的定义。该变量可以是一个标识符（例如 int a 中的 a），也可以是一个标识符后面跟着若干对方括号括起来的数字（例如 int a [10] [2] 中的 a[10][2]，这种情况下 a 是一个数组）。

2) FunDec 表示对一个函数头的定义。它包括一个表示函数名的标识符以及由一对圆括号括起来的形参列表，该列表由 VarList 表示（也可以为空）。VarList 包括一个或多个 ParamDec，其中每个 ParamDec 都是对一个形参的定义，该定义由类型描述符 Specifier 和变量定义 VarDec 组成。例如一个完整的函数头为：foo (int x, float y[10])。

### A. 2. 5 Statements

这一部分的产生式主要与语句有关：

1) CompSt 表示一个由一对花括号括起来的语句块。该语句块内部先是一系列的变量定义 DefList, 然后是一系列的语句 StmtList。可以发现, 对 CompSt 这样的定义, 是不允许在程序的任意位置定义变量的, 必须在每一个语句块的开头才可以定义。

2) StmtList 就是零个或多个 Stmt 的组合。每个 Stmt 都表示一条语句, 该语句可以是一个在末尾添了分号的表达式 (Exp SEMI), 可以是另一个语句块 (CompSt), 可以是一条返回语句 (RETURN Exp SEMI), 可以是一条 if 语句 (IF LP Exp RP Stmt), 可以是一条 if-else 语句 (IF LP Exp RP Stmt ELSE Stmt), 也可以是一条 while 语句 (WHILE LP Exp RP Stmt)。

## A.2.6 Local Definitions

这一部分的产生式主要与局部变量的定义有关:

1) DefList 这个语法单元在前面曾出现在 CompSt 以及 StructSpecifier 产生式的右边, 它就是一串像 int a; float b, c; int d [10]; 这样的变量定义。一个 DefList 可以由零个或者多个 Def 组成。

2) 每个 Def 就是一条变量定义, 它包括一个类型描述符 Speicifier 以及一个 Declist, 例如 int a, b, c。由于 Declist 中的每个 Dec 又可以变成 VarDec ASSIGNOP Exp, 这允许我们对局部变量在定义时进行初始化, 例如 int a=5。

## A.2.7 Expressions

这一部分的产生式主要与表达式有关:

1) 表达式演化出的形式可以多种多样, 但总体上看不外乎下面几种:

a) 包含二元运算符的表达式: 赋值表达式 (Exp ASSIGNOP Exp)、逻辑与 (Exp AND Exp)、逻辑或 (Exp OR Exp)、关系表达式 (Exp RELOP Exp) 以及四则运算表达式 (Exp PLUS Exp 等)。

b) 包含一元运算符的表达式: 括号表达式 (LP Exp RP)、取负 (MINUS Exp) 以及逻辑非 (NOT Exp)。

c) 不包含运算符但又比较特殊的表达式：函数调用表达式（带参数的 ID LP Args RP 以及不带参数的 ID LRP）、数组访问表达式（Exp LB Exp RB）以及结构体访问表达式（Exp DOT ID）。

d) 最基本的表达式：整型常数（INT）、浮点型常数（FLOAT）以及普通变量（ID）。

2) 语法单元 Args 表示实参列表，每个实参都可以变成一个表达式 Exp。

3) 由于表达式中可以包含各种各样的运算符，为了消除潜在的二义性问题，我们需要给出这些运算符的**优先级（precedence）**以及**结合性（associativity）**，如表 A-1 所示。

表 A—1 C--语言中运算符的优先级和结合性

优先级 <sup>①</sup>	运算符	结合性	描述
1	(, )	左结合	括号或函数调用
	[, ]		数组访问
	.		结构体访问
2	-	右结合	取负
	!		逻辑非
3	*	左结合	乘
	/		除
4	+		加
	-		减
5 <sup>②</sup>	<		小于
	<=		小于或等于
	>		大于
	>=		大于或等于
	==		等于
	!=		不等于
6	&&		逻辑与
7			逻辑或
8	=	右结合	赋值

①数值越小表示代表优先级越高。

②在标准 C 语言中，“>”、“<”、“>=”和“<=”四个关系运算符的优先级要比“==”和“!=”两个运算符的优先级高；在这里，我们为了方便处理而将它们的优先级统一了。

### A. 2. 8 Comments

C--源代码可以使用两种风格的注释：一种是使用双斜线“//”进行单行注释，在这种情况下，该行在“//”符号之后的所有字符都将作为注释内容而直接被词法分析

程序丢弃掉；另一种是使用“/\*”以及“\*/”进行多行注释，在这种情况下，在“/\*”与之后最先遇到的“\*/”之间的所有字符都被视作注释内容。需要注意的是，“/\*”与“\*/”是不允许嵌套的：即在任意一对“/\*”和“\*/”之间不能再包含成对的“/\*”和“\*/”，否则编译器需要进行报错。