C-minus扩充介绍

C-minus的语法

1. program → declaration-list
2. declaration-list → declaration-list declaration | declaration
3. declaration → var-declaration | fun-declaration
4. var-declaration → type-specifier **ID ;** | type-specifier **ID** [**NUM**] **;**
5. type-specifier → **int** | **void**
6. fun-declaration → type-specifier **ID** (params) compound-stmt
7. params → param-list | **void**
8. param-list → param-list , param | param
9. param → type-specifier **ID** | type-specifier **ID** []
10. compound-stmt → **{** local-declarations statement-list**}**
11. local-declarations → local-declarations var-declaration | empty
12. statement-list → statement-list statement | empty
13. statement→ expression-stmt
| compound-stmt
| selection-stmt
| iteration-stmt
| return-stmt
14. expression-stmt → expression **;** | **;**
15. selection-stmt→ **if** (expression) statement
| **if** (expression) statement **else** statement
16. iteration-stmt → **while** (expression) statement
17. return-stmt → **return ;** | **return** expression **;**
18. expression → var = expression | simple-expression
19. var → **ID** | **ID** [expression]
20. simple-expression → additive-expression relop additive-expression | additive-expression
21. relop → <= | < | > | >= | == | **!=**
22. additive-expression → additive-expression addop term | term
23. addop → + | **-**
24. term → term mulop factor | factor
25. mulop → ***** | /
26. factor → (expression) | var | call | **NUM**
27. call → **ID** (args)
28. args → arg-list | empty
29. arg-list → arg-list **,** expression | expression

C-minus的语义

在上述语法规则中，我们定义了C-minus语言的语法，接着，我们对照语法规则，给出相关的语义和解释。

1. program → declaration-list
2. declaration-list → declaration-list declaration | declaration
3. declaration → var-declaration | fun-declaration

一个`程序`由一系列`声明`组成，声明包括了`函数声明`与`变量声明`，它们可以以任意顺序排列。

所有的变量和函数必须在使用前先进行声明

一个`程序`中至少要有一个`声明`，且最后一个`声明`必须是`void main(void)`形式的`函数声明`。

因为没有原型这个概念，`C-`不区分声明和定义。

4. var-declaration → type-specifier **ID ;** | type-specifier **ID** [**NUM**] **;**
5. type-specifier → **int** | **void**

`C-`的基础类型只有整型和`void`。而在变量声明中，只有整型可以使用，`void`仅用于函数声明。

一个`变量声明`定义一个整型(int)的变量或者一个整型数组变量（这里整型指的是32位有符号整型）。

数组变量在声明时，**NUM**应当大于0。

一次只能声明一个变量。

6. fun-declaration → type-specifier **ID** (params) compound-stmt
7. params → param-list | **void**
8. param-list → param-list , param | param
9. param → type-specifier **ID** | type-specifier **ID** []

`函数声明`包含了返回类型，标识符，由逗号分隔的`形参`列表，还有一个`复合语句`。

当函数的返回类型是`void`时，函数不返回任何值。

函数的参数可以是`void`，也可以是一个列表。当函数的`形参`是`void`时，调用该函数时不用传入任何参数。

`形参`中跟着中括号代表数组参数，它们可以有不同长度。

整型参数通过值来传入函数（pass by value），而数组参数通过引用来传入函数（pass by reference，即指针）。

函数的`形参`拥有和`函数声明`的`复合语句`相同的作用域，并且每次函数调用都会产生一组独立内存的参数。（和c语言一致）

函数可以递归调用。

10. compound-stmt → { local-declarations statement-list}

一个`复合语句`由一对大括号和其中的`局部声明`与`语句列表`组成

`复合语句`的执行时，对包含着的语句按照`语句列表`中的顺序执行

`局部声明`拥有和`复合语句`中的`语句列表`一样的作用域，且其优先级高于任何同名的全局声明（常见的静态作用域）

11. local-declarations → local-declarations var-declaration | empty
12. statement-list → statement-list statement | empty

`局部声明`和`语句列表`都可以为空（empty表示空字符串，即ε）

13. statement→ expression-stmt
    | compound-stmt
    | selection-stmt
    | iteration-stmt
    | return-stmt
14. expression-stmt → expression **;** | **;**

`表达式语句`由一个可选的`表达式`（即可以没有`表达式`）和一个分号组成

我们通常使用表达式语句中的表达式计算时产生的副作用，所以这种语句用于赋值和函数调用

15. selection-stmt→ **if** (expression) statement
| **if** (expression) statement **else** statement

if语句中的表达式将被求值，若结果为0，则第二个语句执行（如果存在的话），否则第一个语句会执行。

为了避免歧义，**else**将会匹配最近的**if**

16. iteration-stmt → **while** (expression) statement

while语句是C-中唯一的迭代语句。它执行时，会不断对表达式进行求值，并且在对表达式的求值结果为0前，循环执行执下面的语句

17. return-stmt → **return ;** | **return** expression **;**

return语句可以返回值，也可以不返回值。

未声明为**void**类型的函数必须返回整型值。

return会将程序的控制转移给当前函数的调用者，而**main**函数的return会使得程序终止

18. expression → var = expression | simple-expression
19. var → **ID** | **ID** [expression]

一个表达式可以是一个变量引用（即var）接着一个赋值符号（=）以及一个表达式，也可以是一个简单表达式。

var可以是一个整型变量，或者一个取了下标的数组变量。

一个负的下标会导致程序终止，但是对于上界并不做检查。

赋值语义为：先找到var代表的变量地址（如果是数组，需要先对下标表达式求值），然后对右侧的表达式进行求值，求值结果将存储在先前找到的地址中。同时，该值将作为赋值表达式的求值结果。

在C中，赋值对象（即var）必须是左值，而左值可以通过多种方式获得。C-中，唯一的左值就是通过var的语法得到的，因此C-通过语法限制了var为左值，而不是像C中一样通过类型检查，这也是为什么C-中不允许进行指针算数。

20. simple-expression → additive-expression relop additive-expression | additive-expression
21. relop → <= | < | > | >= | == | !=

一个简单表达式是一个加法表达式或者两个加法表达式的关系运算。当它是加法表达式时，它的值就是加法表达式的值。而当它是关系运算时，如果关系运算结果为真则值为1，反之则值为0。

22. additive-expression → additive-expression addop term | term
23. addop → + | -
24. term → term mulop factor | factor
25. mulop → * | /

加法表达式表现出了四则运算的结合性质与优先级顺序，四则运算的含义和C中的整型运算一致。

26. factor → (expression) | var | call | **NUM**

因数可以是一个括号包围的表达式（此时它的值是表达式的值），或者是一个变量（此时它的值是变量的值），或者是一个函数调用（此时它的值是函数调用的返回值），或者是一个数字字面量（此时它的值为该字面量的值）。当因数是数组变量时，除非此时它被用作一个函数调用中的数组参数，否则它必须要带有下标。

27. call → **ID** (args)
28. args → arg-list | empty

29. arg-list → arg-list , expression | expression

函数调用由一个函数的标识符与一组括号包围的实参组成。实参可以为空，也可以是由逗号分隔的的表达式组成的列表，这些表达式代表着函数调用时，传给形参的值。函数调用时的实参数量和类型必须与函数声明中的形参完全一致。

C-中包含两个预定义的函数input与output，它们的声明为:

```
int input(void) {...}
void output(int x) {...}
```

input函数没有形参，且返回一个从标准输入中读到的整型值。output函数接受一个整型参数，然后将它的值打印到标准输出，并输出换行符。

除此之外，其它规则和C中类似，比如同一个作用域下不允许定义重名变量或函数

惯用词法

1. 下面是语言的关键字:

```
else if int return void while
```

所有的关键字都是保留字，并且必须是小写。

2. 下面是专用符号:

```
**+ - \* / < <= > >= == != = ; , ( ) [ ] { } /\* \*/**
```

3. 其他标记是**ID**和**NUM**，通过下列正则表达式定义:

```
ID = letter letter\*

NUM = digit digit\*

letter = a|..|z|A|..|Z

digit = 0|..|9
```

小写和大写字母是有区别的。

4. 空格由空白、换行符和制表符组成。空格通常被忽略，除了它必须分开**ID**、**NUM**关键字。
5. 注释用通常的C语言符号 `/* . . . */` 围起来。注释可以放在任何空白出现的位置(即注释不能放在标记内)上，且可以超过一行。注释不能嵌套。