程序定义：

$< program >\rightarrow < segment >< program > |\epsilon$

> program表示主程序，segment表示语句片段
>
> segment主要包括变量声明定义和函数声明定义等
>
> 声明用extern（主要用于外部声明）

程序段定义：

$< segment >\rightarrow extern < type >< def > | < type >< def >$

$< type >\rightarrow int|char|void$

> type表示基本类型，如果不定义则默认为int
>
> def表示定义

变量定义：

$< vardef >\rightarrow id < norvardef > | *id < init >$

$< norvardef >\rightarrow [num] | \ < init >$

$< init >\rightarrow \ =< expr > |\epsilon$

$< varlist >\rightarrow , < vardef >< varlist > |;$

> vardef表示变量定义，区分了普通变量与指针变量定义
>
> norvardef表示数组定义or变量初始化，这里不支持数组初始化
>
> init表示初始化，expr表示表达式，具体放在后文来说
>
> varlist表示支持同时定义多个变量，可以用表示变量定义

函数定义：

函数定义：

$< fundef >\rightarrow id(< para >) < funtail >$

$< funtail >\rightarrow ; | < funbody >$

> fundef表示函数：函数名(参数列表) 函数声明/定义
>
> funtail是表示函数为声明还是定义

参数定义：

$< para >\rightarrow < type >< paradef >< paralist > |\epsilon$

$< paradef >\rightarrow \ *id|id < paratail >$

$< paratail >\rightarrow \ [num]|\epsilon$

$< paralist > \rightarrow \ , < type >< paradef >< paralist > | \epsilon$

> para表示参数的总定义
>
> paradef表示参数是指针or变量(数组)
>
> paratail表示是否为数组
>
> parallist表示定义多个参数

函数体定义:

$< funbody > \rightarrow \{< funprogram >\}$

$< funprogram > \rightarrow < localdef >< funprogram > | < statement >< funprogram > | \epsilon$

$< localdef > \rightarrow < type >< vardef >< varlist >$

> funbody表示函数体定义
>
> funprogram表示函数的程序，可以有局部变量定义，也可以有语句
>
> localdef就是变量定义，可以用上文的表示
>
> statement放在后文来说

变量总定义：

$< def > \rightarrow \ id < deftail > | ^{*} id < init >< varlist >$

$< deftail > \rightarrow < norvardef >< varlist > | (< para >) < funtail >$

> 为什么不能直接 -> | 呢?
>
> 因为这里涉及左公因子id，因此需要展开后进行拆分

表达式定义：

运算符	含　义	优先级	结合性
=	赋值	10	右结合
||	逻辑或	9	左结合
&&	逻辑与	8	左结合
> < >= <= == !=	大于、小于、大于等于、小于等于、等于、不等于	7	左结合
+ -	加法、减法	6	左结合
* / %	乘法、除法、取模	5	左结合
! - & * ++ --	逻辑非、取负、取址、指针、前置++、前置--	4	右结合
++ --	后置++、后置--	3	右结合
()	括号	2	左结合
[] ()	数组索引、函数调用	1	左结合

构造表达式语法时，需要从运算符优先级从低到到来进行考虑。(上表中，优先级数字越低表示优先级越高)

$< assexpr > \rightarrow < oorexpr >< asstail >$

$< asstail > \rightarrow \ \ =< orrexpr >< asstail > | \epsilon$

> =
>
> 值得注意的是，多次赋值：a = b= c= d 这样的式子应该从右到左运算

$< oorexpr >\rightarrow< aandexpr >< oortail >$

$< oortail >\rightarrow \ || < aandexpr >< oortail > |\epsilon$

> ||

$< aandexpr >\rightarrow< orexpr >< aandtail >$

$< aandtail >\rightarrow \ \&\& < orexpr >< aandtail > |\epsilon$

> &&

$< orexpr >\rightarrow< xorexpr >< ortail >$

$< ortail >\rightarrow \ or < xorexpr >< ortail > |\epsilon$

> |

$< xorexpr >\rightarrow< andexpr >< xortail >$

$< xortail >\rightarrow \ xor < xorexpr >< xortail > |\epsilon$

> ^

$< andexpr >\rightarrow< cmpexpr >< andtail >$

$< andtail >\rightarrow \ and < cmpexpr >< andtail > |\epsilon$

> &

$< cmpexpr >\rightarrow< aloexpr >< cmptail >$

$< cmptail >\rightarrow \ \ < cmps >< aloexpr >< cmptail > |\epsilon$

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

> 比较运算符

$< aloexpr >\rightarrow< item >< alotail >$

$< alotail >\rightarrow \ \ < alos >< item >< alotail > |\epsilon$

$< alos >\rightarrow +|-$

> + - 运算符

$< item >\rightarrow< factor >< itemtail >$

$< itemtail >\rightarrow \ \ < its >< factor >< itemtail > |\epsilon$

$< its >\rightarrow *|/|\%$

> * / %运算符

$< factor >\rightarrow< lop >< factor > | < val >$

$< lop >\rightarrow \ !| - |\&|^*| + +| - -| \sim |\epsilon$

> 逻辑非 取负 取址 指针 左自增 左自减

$< val >\rightarrow< element >< rop >$

$< rop >\rightarrow \ \ + +| - -|\epsilon$

> 右自增 右自减

$< element >\rightarrow id < idexpr > | (expr) | literal$

$< idexpr >\rightarrow [< expr >] | (< realarg >)|\epsilon$

$< realarg >\rightarrow < expr >< arglist > |\epsilon$

$< arglist >\rightarrow ,< expr >< arglist > |\epsilon$

> 元素可以是变量，数组，函数，括号表达式以及常量

$< literal >\rightarrow num|char|str$

> 常量包括数字，字符和字符串

$< expr >\rightarrow < assexpr >$

$< altexpr >\rightarrow < expr > |\epsilon$

> 表达式从assexpr开始
>
> altexpr可以让表达式为空，常用语for语句中

语句定义：

语句要分为表达式语句，循环语句，判断语句，break，continue，return以及自定义语句等等。

$< statement >\rightarrow < altexpr >; | < whilestat > | < forstat > | < dowhilestat >$

$< statement >\rightarrow < ifstat > | < switchstat > | < secloudstat > |break; |continue; |return < altexpr >;$

$< statement >\rightarrow < readstat > | < writestat >$

循环语句定义：

$< whilestat >\rightarrow while(< altexpr >) < funbody >$

$< dowhilestat >\rightarrow do < funbody > while(< altexpr >);$

> while和do while形式还是比较清晰的

$< forstat >\rightarrow for(< forinit >;< altexpr >;< altexpr >) < funbody >$

$< forinit >\rightarrow < localdef > | < altexpr >$

> 这里就可以体现出altexpr->expr|e 的简便性了

判断语句定义：

$< ifstat >\rightarrow if(< altexpr >) < funbody >< elsestat >$

$< elsestat >\rightarrow else < funbody > |\epsilon$

> if语句的定义

$< switchstat >\rightarrow switch(< expr >)\{< casestat >\}$

$< casestat >\rightarrow case < literal >:< funprogram >< casestat > |default :< funprogram >$

> switch语句的定义

其他语句定义:

$< secloudstat >\rightarrow secloud(num);$

> secloud(x)是一个自定义函数，随机返回123 57 152 110这四个数
>
> haha just for fun!

$< readstat >\rightarrow read(num);$

$< writestat >\rightarrow write(id);$

> 目前只支持读入数字和输出数字，具体实现以后扩展
>
> 即id的类型要求为int型