以下是对上述问题的详细解答：

### 1. 整数的正规文法和正规式

**正规文法**：

设整数的非终结符为<N>，终结符集合为数字{0-9}，则文法规则如下：

<N> -> <D>（<D>表示一位数字，且不为0开头）

<D> -> 1 | 2 | 3 |... | 9

<N> -> <D><N>

<N> -> 0

**正规式**：(1|2|...|9)(0|1|...|9)\*|0，表示以非零数字开头后面跟任意多个数字或者单独一个数字0的形式，符合不带前缀零的整数定义。

### 2. 变量的正规文法和自动机

**正规文法**：

设变量的非终结符为<V>，终结符集合包含汉字{甲、乙、丙、丁}、字母{a-z, A-Z}、数字{0-9}以及下划线\_，规则如下：

<V> -> <H>（<H>表示以汉字开头的符号串）

<V> -> <L>（<L>表示以字母开头的符号串）

<H> -> 甲<X> | 乙<X> | 丙<X> | 丁<X>（<X>表示后续可跟汉字、字符、数字和下划线构成的符号串）

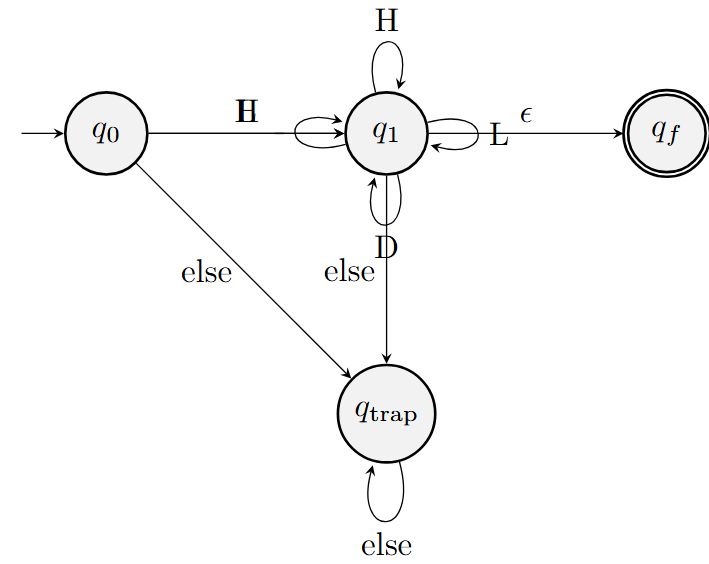
<L> -> a<X> | b<X> |... | z<X> | A<X> | B<X> |... | Z<X>

<X> -> <H> | <L> | <D> | \_<X>（<D>为数字，同整数定义中的数字情况）

<X> -> ε（ε表示空串，即可以结束）

**自动机**：

可以构建一个有限自动机，它有初始状态q0，接受状态qf。从q0出发，若输入字符是汉字或者字母则分别转移到对应状态（处理以汉字或字母开头的情况），之后在后续状态中，对于输入的汉字、字母、数字、下划线等进行相应状态转移，当遇到没有后续可转移情况时（合法的变量结束情况），可以转移到接受状态qf。若在过程中遇到不符合变量定义的字符输入，则进入一个“陷阱”状态，表示不接受该输入串，不是合法的变量。



### 3. 描述表达式的LL(1)文法及对b + c % b的分析

**LL(1)文法**：

设<E>表示表达式，<T>表示项，<F>表示因子，将变量用b表示，整数用c表示，文法规则如下：

<E> -> <T> <E'>

<E'> -> + <T> <E'> | <T> <E'> | ε

<T> -> <F> <T'>

<T'> -> \* <F> <T'> | / <F> <T'> | \ <F> <T'> | ε

<F> -> b | c | (<E>)

**预测分析表构建及b + c % b分析过程**：

**预测分析表构建**：

首先求各非终结符的FIRST集合和FOLLOW集合。例如：

FIRST(<E>) = {b, c, (}

FIRST(<E'>) = {+, -, ε}

FIRST(<T>) = {b, c, (}

FIRST(<T'>) = {\*, /, \, ε}

FIRST(<F>) = {b, c, (}

FOLLOW(<E>) = {), $}（$表示输入结束符）

FOLLOW(<E'>) = {), $}

FOLLOW(<T>) = {+, -, ), $}

FOLLOW(<T'>) = {+, -, ), $}

FOLLOW(<F>) = {\*, /, \, +, -, ), $}

根据FIRST和FOLLOW集合来构建预测分析表，表中元素是相应的产生式规则。比如在M[<E>, b]处填入<E> -> <T> <E'>等。

**分析过程**：

输入串为b + c % b，分析栈初始为$<E>（$在栈底）。

步骤如下：

栈顶为<E>，输入符号为b，查预测分析表得<E> -> <T> <E'>，将栈顶替换为<E'><T>。

栈顶为<T>，输入符号为b，查分析表得<T> -> <F> <T'>，栈顶替换为<T'><F>。

栈顶为<F>，输入符号为b，得<F> -> b，将栈顶b弹出，输入符号也读入下一个（此时为+）。

栈顶为<T'>，输入符号为+，查表得<T'> -> ε，将栈顶<T'>弹出。

栈顶为<E'>，输入符号为+，查表得<E'> -> + <T> <E'>，将栈顶替换为<E'><T>+，输入符号读入下一个（为c）。

以此类推，按照分析表不断进行栈的替换和输入符号的读取，直到栈中只剩$，表示分析成功，输入串是该文法的合法表达式。

### 4. 描述赋值语句的文法及语法分析

**赋值语句的文法**：

设<S>表示赋值语句，<E>表示表达式（同上述表达式定义），文法规则为：

<S> -> b = <E>

文法类型判断：SLR（1）

对于这个简单的赋值语句文法，它是LR(0)文法。因为它不存在移进 归约冲突和归约 归约冲突，状态机中每个状态下的项目集对于面临的输入符号动作都是明确的（要么移进，要么归约，要么接受），满足LR(0)文法的要求。同时它也属于SLR(1)、LR(1)、LALR(1)，因为LR(0)是这些更复杂的LR分析方法的一个基础子集情况，当满足LR(0)时必然也满足其他几种在该简单文法下的分析要求（不过从严格定义角度它们有各自的分析表构建和冲突判断等细节差异）。

**自底向上对b = b + c % b的分析过程（以LR(0)分析为例简单说明思路）**：

首先构建该文法的LR(0)项目集规范族和分析表（这里构建过程略，和常规LR(0)分析类似）。

分析栈初始为$（栈底），输入串为b = b + c % b。

不断根据分析表进行动作：

开始时，读入b，按照分析表执行移进操作，栈变为$b，输入串变为= b + c % b。

接着读入=，执行移进，栈变为$b =，输入串变为b + c % b。

再读入b，移进，栈变为$b = b，输入串变为+ c % b。

之后随着不断读入符号，按照分析表进行移进和归约操作（例如当遇到符合表达式归约条件时进行归约等），最终当栈变为$<S>且输入串读完（只剩$），表示分析成功，该输入串是符合赋值语句文法的合法语句。

以上完整地解答了关于易语言相关语法描述的各部分问题，在实际应用中可以通过编写程序等方式更详细地实现这些语法分析过程展示。

|  |  |
| --- | --- |
| 自动机（状态转移示意，简略描述核心逻辑） |  |
| 状态 | 输入（汉字） |
| *q*0​（初始） | 转移到处理汉字后续状态 |
| 处理汉字后续状态 | 转移到自身等（继续处理后续符号） |
| 处理字母后续状态 | 转移到“非法”状态 |
| 接受状态 *qf*​ | 无（已接受，结束） |

|  |  |
| --- | --- |
| 预测分析表（部分示意，完整需计算各非终结符的 FIRST 和 FOLLOW 集合） |  |
| 非终结符 | b*b* |
| ---- | ---- |
| E*E* | E→TE′*E*→*TE*′ |
| E′*E*′ |  |
| T*T* | T→FT′*T*→*FT*′ |
| T′*T*′ |  |
| F*F* | F→b*F*→*b* |
| b + c % b 分析过程 |  |
| 步骤 | 分析栈（栈底在左） |
| ---- | ---- |
| 1 | E*E* |
| 2 | E′T*E*′*T* |
| 3 | E′T′F*E*′*T*′*F* |
| 4 | E′T′*E*′*T*′ |
| 5 | E′*E*′ |
| 6 | E′T*E*′*T* |
| 7 | E′T′F*E*′*T*′*F* |
| 8 | E′T′*E*′*T*′ |
| 9 | E′*E*′ |

|  |  |
| --- | --- |
| 自底向上对 b = b + c % b 的分析过程（以 LR(0)*LR*(0) 分析简单示意思路，实际需构建完整分析表等） |  |
| 步骤 | 分析栈（栈底在左） |
| ---- | ---- |
| 1 |  |
| 2 | b*b* |
| 3 | b=*b*= |
| 4 | b=b*b*=*b* |