1. 中缀逻辑表达式-a*(b+c)<d+e AND t。

(1). 写出等价的逆波兰后缀表示；

(2). 写出四元式表示。

答案：

(1)a@bc+*de+<tAND

(2)

(@,a,-,T1)

(+,b,c,T2)

(*,T1,T2,T3)

(+,d,e,T4)

(<,T3,T4,T5)

(AND,T5,t,T6)

2.有如下表达式

-(a+(b-c))+d

分别写出其等价的逆波兰表示（后缀式）和四元式表示。

答案：

逆波兰表示为 abc-+@d+

四元式表示为:

(1) (-,b, c, T1)

(2) (+, a, T1,T2)

(3) (-, T2, -,T3)

(4) (+, T3, d,T4)

3. 将下列语句翻译为逆波兰表示（后缀式），三元式和间接三元式序列和四元式表示：

a:=(b+c)*e+(b+c)/f

答案：

逆波兰表示为：bc+e*bc+f/+:=

三元式序列为:

(1) (+, b, c)

(2) (*,(1),e)

(3) (+, b, c)

(4) (/,(3),f)

(5) (+,(2),(4))

(6) (:=,a,(5))

间接三元式表示为：

三元式表	间接码表
(1) (+, b, c)	(1)
(2) (*,□, e)	(2)
(3) (/,□, f)	(1)
(4) (+,□,□)	(3)
(5) (:=,a,□)	(4)
	(5)

四元式表示为:

(1) (+, b, c, T1)
(2) (*, T1,e, T2)
(3) (+, b, c, T3)
(4) (/, T3, f, T4)
(5) (+, T2,T4, T5)
(6) (:=,T5, -, a)

4. 现有文法 G_1、G_2 如下：欲将 G_1 定义的 expression 转换如 G_2 的 E 所描述的形式。给出其语法制导翻译的语义描述。（提示：可采用类似 yacc 源程序的形式，所涉及的语义函数须用自然语言给予说明，不用抄写产生式，用产生式编号表示。）

G_1

⑴ <PROGRAM>→<decl statement>;BEGIN<statement list>END
⑵<decl statement>→VAR<iddecl>
⑶<iddecl>→<iddecl>,id:<type decl>
⑷<iddecl>→id:<type decl>
⑸<type decl>→int
⑹<type decl>→bool
⑺<statement list>→<expression>
⑻<statement list>→<statement list>;<expression>
⑼<expression>→<expression>AND<expression>
⑽<expression>→<expression>*<expression>
⑾<expression>→id
⑿<expression>→NUM
⒀<expression>→true
⒁<expression>→false

要求：⑴ <expression>中的 id 必须在<decl statement>中先声明。
⑵ AND 和*分号是常规的布尔和算术运算符，要求其运算对象的相应类型匹配。

G_2： E→EE* | EEand | id | num | true | false

答案：

从题目分析, 翻译的关键是把 G1 描述的中缀形式的表达式转换为 G2 描述的后缀形式的表达式, 并要进行一定的类型检查。 在下面的语义动作中, 采用类似 yacc 源程序的形式, $$代表相应产生式的左部符号, $1 代表产生式右部的第一个文法符号, $2 代表产生式右部的第二个文法符号, 依此类推。 在下面语义动作中, lookup(name)的功能是对 name 查找符号表并返回其类型,如果查不到则报错; lexeme(token)给出 token 的词法值。

⑴ <PROGRAM>→<decl statement>;BEGIN<statement list>END
{ $$.code := $4.code;}
⑵<decl statement>→VAR<iddecl>
{ }
⑶<iddecl>→<iddecl>,id:<type decl>
{ enterid($3,$5.type);}
⑷<iddecl>→id:<type decl>
{ enterid($1,$3.type);}

⑸<type decl>→int
{ $$.type := int ;}
⑹<type decl>→bool
{ $$.type := bool;}
⑺<statement list>→<expression>
{ $$.code := $1.code;}
⑻<statement list>→<statement list>;<expression>
{ $$.code := $1.code ‖ $3.code;}
⑼<expression>→<expression>AND<expression>
{ if $1.type = bool and $3.type = bool
then $$.type := bool
else typeerror;
$$.code := $1.code ‖$3.code ‖'and' ; }
⑽<expression>→<expression>*<expression>
{ if $1.type = int and $3.type = int
then $$.type := int
else typeerror;
$$.code := $1.code ‖$3.code ‖'*' ; }
⑾<expression>→id
{ $$.type := lookup($1);
$$.code := lexeme($1); }
⑿<expression>→NUM
{ $$.type := int;
$$.code := lexeme($1); }
⒀<expression>→true
{ $$.type := bool;
$$.code := lexeme($1); }
⒁<expression>→false
{ $$.type := bool;
$$.code := lexeme($1); }