Generación de Código

1. Suponga una sentencia que calcula el promedio de una lista de expresiones y se lo asigna a un identificador y cuya sintaxis está representada con las siguiente reglas gramaticales:

   A -> id := P
   P -> prom (L)
   L -> L , E | E
   E -> E + T | E – T | T
   T -> T * F | T / F | F
   F -> id | cte | (E)

   a. Representar la sentencia **a:= prom ([a+b, 3, c*(d-a)])** en polaca inversa de manera que toda la semántica sea resuelta en la notación intermedia

Algoritmo Promedio

$$\frac{(a_n+b_n+...+w_n+z_n)}{n}$$

a	:=	p	(	[	a	+	b	,	3	,	c	*	(	d	-	a	)	]	)
Árbol																			
	-=																		
id												/							
										+				@n					
								+				*							
						+			cte		id				-				
					id		id							id		id			

Reglas
R11, R10, R07, R11, R10, R05, R04, R12, R10, R07, R03, R11, R10, R11, R10, R07, R11, R10, R06, R13, R08, R07, R03, R02, R01, R00

Polaca (operando, operando, operador)																		
a	b	+	3	+	c	d	a	-	*	+	@a	=	@a	@n	/	id	@a	=
R11 R10 R07	R11 R10	R05 R04	R12 R10 R07	R03	R11 R10	R11 R10 R07	R11 R10	R06 R13	R08 R07	R03	R02					R01 OK		

   b. Escribir las acciones semánticas en cada regla para generar código en polaca inversa para cualquier sentencia con el formato indicado.

R00 A´-> A	{ generar_(polaca) }
R01 A -> id := P	{ insertar_pol(@acum); insertar_pol(id); insertar_pol(=) }
R02 P -> prom (L)	{ insertar_pol(@acum); insertar_pol(=); insertar_pol(@acum); insertar_pol(@n),insertar_pol(/) }
R03 L -> L , E	{ insertar_pol(+);@n++ }
R04 L -> E	{ @n=1 }
R05 E -> E + T	{ insertar_pol(+) }
R06 E -> E – T	{ insertar_pol(-) }
R07 E -> T	
R08 T -> T * F	{ insertar_pol(*) }
R09 T -> T / F	{ insertar_pol(/) }
R10 T -> F	
R11 F -> id	{ insertar_pol(id) }
R12 F -> cte	{ insertar_pol(cte) }
R13 F -> (E)	

Test Bison	
insertar_pol(a) insertar_pol(b) insertar_pol(+) insertar_pol(3) insertar_pol(+) insertar_pol(c) insertar_pol(d) insertar_pol(a) insertar_pol(-)	insertar_pol(*) insertar_pol(+) insertar_pol(@acum) insertar_pol(=) insertar_pol(3) insertar_pol(/) insertar_pol(a) insertar_pol(=) generar_(polaca) OK

2. Para la sentencia del ejercicio anterior ,
   a. Representar la sentencia **a:= prom ([a+b, 3, c*(d-a)])** en árbol sintáctico de manera que toda la semántica sea resuelta en la notación intermedia

a	:=	p	(	[	a	+	b	,	3	,	c	*	(	d	-	a	)	]	)
Árbol																			
	-=																		
id												/							
										+				n					
								+				*							
						+			cte		id				-				
					id		id							id		id			

   b. Escribir las acciones semánticas en cada regla para generar código en árbol sintáctico para cualquier sentencia con el formato indicado.

R00 A´-> A	Sp=Ap
R01 A -> id := P	Ap=crearNodo(“=”,crearHoja(id),Pp);
R02 P -> prom (L)	Pp=crearNodo(“/”,Lp,crearHoja(@n));
R03 L -> L , E	Lp=crearNodo(“+”,Lp,Ep); n++;
R04 L -> E	Lp=Ep;n=1;
R05 E -> E + T	Ep=crearNodo(“+”,Ep,Tp);

R06 E -> E – T	Ep=crearNodo(“-”,Ep,Tp);
R07 E -> T	Ep=Tp;
R08 T -> T * F	Tp=crearNodo(“*”,Tp,Fp);
R09 T -> T / F	Tp=crearNodo(“/”,Tp,Fp);
R10 T -> F	Tp=Fp;
R11 F -> id	Fp=crearHoja(id);
R12 F -> cte	Fp=crearHoja(cte);
R13 F -> (E)	Fp=Ep;

Test Bison y Representación	
Fp = crearHoja(a) Tp = Fp Ep = Tp Fp = crearHoja(b) Tp = Fp Ep = crearNodo('+', Ep, Tp) Lp = Ep Fp = crearHoja(3) Tp = Fp Ep = Tp Lp = crearNodo('+', Lp, Ep) Fp = crearHoja(c) Tp = Fp	Fp = crearHoja(d) Tp = Fp Ep = Tp Fp = crearHoja(a) Tp = Fp Ep = crearNodo('-', Ep, Tp) Fp = Ep Tp = crearNodo('*', Tp, Fp) Ep = Tp Lp = crearNodo('+', Lp, Ep) Pp = crearNodo('/', Lp, crearHoja(3)) Ap = crearNodo(':=', crearHoja(a), Pp) Sp = Ap OK

3. Para la sentencia del ejercicio anterior ,
   a. Representar la sentencia **a:= prom ([a+b, 3, c*(d-a)])** en tercetos de manera que toda la semántica sea resuelta en la notación intermedia

a	:=	p	(	[	a	+	b	,	3	,	c	*	(	d	-	a	)	]	)
Árbol																			
	-=																		
id												/							
										+				n					
								+				*							
						+			cte		id				-				
					id		id							id		id			

   b. Escribir las acciones semánticas en cada regla para generar código en tercetos para cualquier sentencia con el formato indicado

R00 A´-> A	Ai=Ap
R01 A -> id := P	Ai=crearTerceto(“=”,crearTerceto(id),Pi)
R02 P -> prom (L)	Pi=crearTerceto(“/”,Li,crearTerceto(@n))
R03 L -> L , E	Li=crearTerceto(“+”,Li,Ei);n++
R04 L -> E	Li=Ei;n=1
R05 E -> E + T	Ei=crearTerceto(“+”,Ei,Ti)
R06 E -> E – T	Ei=crearTerceto(“-”,Ei,Ti)
R07 E -> T	Ei=Tp
R08 T -> T * F	Ti=crearTerceto(“*”,Ti,Fi)
R09 T -> T / F	Ti=crearTerceto(“/”,Ti,Fi)

R10 T -> F	Ti=Fi
R11 F -> id	Fi=crearTerceto(id)
R12 F -> cte	Fi=crearTerceto(cte)
R13 F -> (E)	Fi=Ei

Test Bison y Representación	
Fi=crearTerceto(a) Ti = Fi Ei = Ti Fi=crearTerceto(b) Ti = Fi Ei = crearTerceto('+', Ei, Ti) Li = Ei Fi = crearTerceto(3) Ti = Fi Ei = Ti Li = crearTerceto('+', Li, Ei) Fi=crearTerceto(c) Ti = Fi	Fi=crearTerceto(d) Ti = Fi Ei = Ti Fi=crearTerceto(a) Ti = Fi Ei = crearTerceto('-', Ei, Ti) Fi = Ei Ti = crearTerceto('*', Ti, Fi) Ei = Ti Li = crearTerceto('+', Li, Ei) Pi = crearTerceto('/', Li, crearTerceto(3)) Ai = crearTerceto(':=', crearTerceto(a), Pi) Ap = Ai OK

4. Sea la gramática del ejercicio 9 de la práctica 1 que resuelve la asignación múltiple.
   a. Representar la sentencia actual:=promedio:=contador:= promedio/ 342 + (contador*contador); en **polaca inversa** de manera que toda la semántica sea resuelta en la notación intermedia

Gramatica
S -> P P -> id = P |d = E ; E -> E + T | E – T | T T -> T * F | T / F | F F -> id | cte | (E)

actual:=promedio:=contador:= promedio/ 342 + (contador*contador);

id	=	id	=	id	=	id	/	cte	+	(	id	*	id	)	;
Árbol															
	=														
id		P													
			=												
		id		P											
					=										
				id		E									;
									+						
						E						T			
						T						F			
							/			(		E		)	
						T		F				F			
						F		cte				T			
						id						*			
											T		F		
											F		cte		
											id				

R00 S -> P	{ generar_(polaca) }
R01 P -> id = P	{ insertar_pol(=) }
R02 P -> id = E ;	{ insertar_pol(=) }
R03 E -> E + T	{ insertar_pol(+) }
R04 E -> E – T	{ insertar_pol(-) }
R05 E -> T	
R06 T -> T * F	{ insertar_pol(*) }
R07 T -> T / F	{ insertar_pol(/) }
R08 T -> F	
R09 F -> id	{ insertar_pol(id) }
R10 F -> cte	{ insertar_pol(cte) }
R11 F -> (E)	

b. Escribir las acciones semánticas en cada regla para generar código en polaca inversa para cualquier sentencia con el formato indicado.

id	id	id	id	cte	/	id	id	*	+	=	=	=
Reglas												
R09	R09	R09	R09 R08	R09	R07 R05	R09 R08	R09	R06	R03	R02	R01	R01 OK

Test Bison	
insertar_pol(actual) insertar_pol(promedio) insertar_pol(contador) insertar_pol(promedio) insertar_pol(342) insertar_pol(/) insertar_pol(contador)	insertar_pol(contador) insertar_pol(*) insertar_pol(+) insertar_pol(=) insertar_pol(=) insertar_pol(=) OK

5. Sea la gramática del ejercicio 9 de la práctica 1 que resuelve la asignación múltiple.
   a. Representar la sentencia **actual:=promedio:=contador:= promedio/ 342 + (contador*contador);** en **árbol sintáctico** de manera que toda la semántica sea resuelta en la notación intermedia

actual:=promedio:=contador:= promedio/ 342 + (contador*contador);

id	=	id	=	id	=	id	/	cte	+	(	id	*	id	)	;
Árbol															
	=														
id		P													
			=												
		id		P											
					=										
				id		E									;
									+						
						E						T			
						T						F			
							/			(		E		)	

						T		F				F			
						F		cte				T			
						id						*			
											T		F		
											F		cte		
											id				

b. Escribir las acciones semánticas en cada regla para generar código en árbol sintáctico para cualquier sentencia con el formato indicado.

R00 S -> P	Sp=Pp
R01 P -> id = P	Pp=crearNodo(crearHoja(id),"=",Pp);
R02 P -> id = E ;	Pp=crearNodo(crearHoja(id),"=",Ep);
R03 E -> E + T	Ep=crearNodo("+",Ep,Tp);
R04 E -> E – T	Ep=crearNodo("-",Ep,Tp);
R05 E -> T	Ep=Tp;
R06 T -> T * F	Tp=crearNodo("*",Tp,Fp);
R07 T -> T / F	Tp=crearNodo("/",Tp,Fp);
R08 T -> F	Tp=Fp;
R09 F -> id	Fp=crearHoja(id);
R10 F -> cte	Fp=crearHoja(cte);
R11 F -> (E)	Fp=Ep;

6. Suponga la siguiente gramática que representa la sintaxis de un lenguaje que solo permite que sus programas tengan sentencias de selección.

```
START -> PROGRAMA
PROGRAMA -> PROGRAMA SENT
PROGRAMA -> SENT
SENT -> SEL | ASIG
ASIG -> ID := EXP
SEL -> IF COND THEN PROGRAMA ENDIF
SEL -> IF COND THEN PROGRAMA ELSE PROGRAMA ENDIF
COND -> IF < CTE
EXP -> EXP + TERM
EXP -> TERM
TERM -> TERM * FACTOR
TERM -> FACTOR
FACTOR -> CTE
FACTOR -> ID
```

a. Representar la siguiente sentencia en polaca inversa de manera que toda la semántica sea resuelta en la notación intermedia

```
IF a < 3 THEN
       b:= c+1
       a:= 28
ELSE
       IF b < 245 THEN
              a:= c+ 67 * b
       ENDIF
ENDIF
```

IF	IF ELSE	WHILE
li ld op cmp bgx xx sentencias_then xx	li ld op cmp bgx xx sentencias_then bi zz sentencias_else	zz li ld op cmp bgx xx sentencias bi zz

01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
a	3	cmp	bge	16	c	1	+	b	=	28	a	=	bi	28	b	245	cmp	bge	28
				|---	----	----	----	----	----	----	----	----	----	--->					
														|---	----	----	----	----	----
																			|---
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
67	b	*	c	+	a	=													
----	----	----	----	----	----	--->													
----	----	----	----	----	----	--->													

b. Escribir las acciones semánticas en cada regla para generar código en **polaca inversa** para cualquier sentencia con el formato indicado.

R00 START-> PRG	
R01 PRG -> PRG STC	
R02 PRG -> STC	
R03 STC -> SLC	
R04 STC -> ASG	
R05 ASG -> id = EXP	pol(id), pol(=)
R06 SLC -> if CND then PRG endif	pol(cmp), pol(bge), apilar(x), avanzar(), PRG y=desapilar(x), x=y
R07 SLC -> if CND then PRG else PRG endif	pol(cmp), pol(bge), apilar(x), avanzar(), PRG, pol(bi), y=desapilar(x), x=y+1, apilar(x), avanzar(), PRG, desapilar(x), x=y
R08 CND -> id < cte	pol(id), pol(cte)
R09 EXP -> EXP + TER	pol(+)
R10 EXP -> TER	
R11 TER -> TER * FAC	pol(*)
R12 TER -> FAC	
R13 FAC -> id	pol(id)
R14 FAC -> cte	pol(cte)

c. Testear con las acciones escritas en el punto b), el resultado del punto a)

Test Bison			
a	1	28	c
3	+	b	67
CMP	=	245	b
BGE	a	CMP	*
16	28	BGE	+
b	=	28	=
c	BI	a	OK

01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
a	3	cmp	bge	16	b	c	1	+	=	a	28	=	bi	28	b	245	cmp	bge	28
				|---	----	----	----	----	----	----	----	----	----	--->					
														|---	----	----	----	----	----
																			|---
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
a	c	67	b	*	+	=													
----	----	----	----	----	----	--->													
----	----	----	----	----	----	--->													

7. Suponga la gramática del ejercicio anterior
   a. Representar el siguiente programa en **tercetos** de manera que toda la semántica sea resuelta en la notación intermedia

```
IF a < 3 THEN
        b:= c+1
        a:= 28
        ELSE
        IF b < 245 THEN
                a:= c+ 67 * b
                ENDIF
ENDIF
```

Tercetos			
a	1	28	c
3	+	b	67
CMP	=	245	b
BGE	a	CMP	*
16	28	BGE	+
b	=	28	=
c	BI	a	OK

   b. Escribir las acciones semánticas en cada regla para generar código en tercetos para cualquier sentencia con el formato indicado.

R00 START-> PRG	Si=Pi
R01 PRG -> PRG STC	Pi=crearTerceto(,Pi,Sti)
R02 PRG -> STC	Pi=Sti
R03 STC -> SLC	Sti=Sli
R04 STC -> ASG	Sti=Ai
R05 ASG -> id = EXP	Ai=crearTerceto(“=”,crearTerceto(id),Ei)
R06 SLC -> if CND then PRG endif	Sli=crearTerceto(“IF”,Ci,Then) Then=crearTerceto(Prgi)
R07 SLC -> if CND then PRG else PRG endif	Sli=crearTerceto(“IF”,Ci,Cpo) Cpo=crearTerceto(,Then,Else) Then=crearTerceto(Prgi) Else=crearTerceto(Prgi)
R08 CND -> id < cte	Ci=crearTerceto(crearTerceto(id),“<”,crearTerceto(cte))
R09 EXP -> EXP + TER	Ei=crearTerceto(“+”,Ei,Ti)
R10 EXP -> TER	Ei=Ti
R11 TER -> TER * FAC	Ti=crearTerceto(“*”,Ti,Fi)

R12 TER -> FAC	**Ti=Fi**
R13 FAC -> id	**Fi=crearTerceto(id)**
R14 FAC -> cte	**Fi=crearTerceto(cte)**

c. Testear con las acciones escritas en el punto b), el resultado del punto a)

8. Suponga la gramática de un lenguaje que solo soporta sentencias de ciclos (del tipo FOR) y sentencias de asignación

START -> PROGRAMA
PROGRAMA -> PROGRAMA SENT
PROGRAMA -> SENT
SENT -> CICLO | ASIG
ASIG -> ID := EXP
CICLO -> FOR INICIO { PROGRAMA } FOREND
INICIO -> ID = CTE TO CTE
EXP -> EXP + TERM
EXP -> TERM
TERM -> TERM * FACTOR
TERM -> FACTOR
FACTOR -> CTE
FACTOR -> ID

a. Representar el siguiente programa en tercetos de manera que toda la semántica sea resuelta en la notación intermedia

```
c:=0
FOR i:=1 TO 20
{
        a:= c+ 67 * b
        b:=b+1
        c:=c+1
}
FOREND
```

b. Escribir las acciones semánticas en cada regla para generar código en tercetos para cualquier sentencia con el formato indicado.

c. Testear con las acciones escritas en el punto b), el resultado del punto a)

9. Suponga la siguiente gramática que representa la sintaxis de un lenguaje que solo permite que sus programas tengan sentencias de ciclos del tipo “while”.

START -> PROGRAMA
PROGRAMA -> PROGRAMA SENT
PROGRAMA -> SENT
SENT -> ASIG | CICLO
CICLO -> WHILE COND PROGRAMA END
COND -> EXP <= EXP
ASIG -> ID := EXP
EXP -> EXP + TERM
EXP -> EXP - TERM
EXP -> TERM

TERM -> TERM * FACTOR
TERM -> FACTOR
FACTOR -> CTE
FACTOR -> ID

a. Representar la siguiente sentencia en polaca inversa de manera que toda la semántica sea resuelta en la notación intermedia

```
WHILE a*3 <= b
        a:= a+2
        b:= 5
        WHILE b <= 5
                a:= a+1
        END
END
```

b. Escribir las acciones semánticas en cada regla para generar código en árbol sintáctico para cualquier sentencia con el formato indicado.

c. Testear con las acciones escritas en el punto b), el resultado del punto a)