Profesor:
Mario Martínez Molina

Tercer Examen Parcial
Compiladores
5 de diciembre de 2018

Ejercicio	1	2	3	4	5	6	7	Total
Puntos	5	5	5	5	30	25	25	100
Calificación	5	0	5	5	5	10	0	30

Preguntas

1. (5 puntos) ¿Qué es un sistema de tipos?
2. (5 puntos) ¿Cuál es la diferencia entre un lenguaje que usa tipado estático y uno que utiliza tipado dinámico?
3. (5 puntos) ¿Qué es un bloque básico?
4. (5 puntos) Mencione y explique 3 optimizaciones que pueden ser realizadas sobre un bloque básico.

Ejercicios

5. (30 puntos) La siguiente SDT calcula el valor decimal de una cadena de 0's y 1's interpretada como un entero binario positivo.

$$B \to B_1 0 \{B.val = 2 \times B_1.val\}$$
$$B \to B_1 1 \{B.val = (2 \times B_1.val) + 1\}$$
$$B \to 1 \{B.val = 1\}$$

(a) Reescriba esta SDT de manera que la gramática subyacente no sea recursiva por la izquierda, al mismo tiempo que el valor del atributo $B.val$ de una cadena binaria completa sea preservado.

(b) Usando su gramática transformada muestre los árboles de parseo anotados para las cadenas binarias `1001` y `110`.

- El alumno no posee los conocimientos sobre árboles sintácticos de manera que pueda aplicarlos a problemas sobre el diseño de un compilador.
- El alumno no posee los conocimientos sobre Definiciones Dirigidas por Sintaxis.
- El no tiene los conocimientos sobre análisis sintáctico y la generación de la representación intermedia.
- El alumno no comprende los fundamentos teóricos sobre la optimización y la generación de código objetivo.

6. (25 puntos) Escriba una SDD que traduzca el siguiente programa a código de tres direcciones. Usando su SDD obtenga el código de tres direcciones equivalente.

```
num[10] a;
num i;
i = 0;
while(i < 10)
{
   a[i] = i;
   i = i + 1;
}
```

1.

Es el conjunto de reglas que establece las operaciones posibles para cada tipo perteneciente al sistema de tipos, para esto cada tipo debe haber sido previamente definido y esta se compone de una serie de valores asociados al mismo. 5 pts

2. En un lenguaje de tipado estático el tipo asociado a cada entidad dentro del lenguaje puede ser determinado en tiempo de compilación mientras que en un lenguaje de tipado dinámico se conoce el tipo asociado de algunas entidades a tiempo de compilación 0 pts

3.

Es un conjunto de instrucciones con la característica de tener una única instrucción líder, esto significa que respetara cierto flujo de datos, esto es útil para el compilador ya que al analizar este flujo puede generar mejoras en el código es decir optimizar IO. 5 pts

9. i) eliminación de subexpresiones

Esta técnica permite omitir código que tiende a repetirse dado que previamente ya fue usada una expresión similar, se consigue analizando el DAG asociado al código de 3 direcciones

5 pts

$a = c + d$ $b = c + d$ $\Rightarrow$ $a = c + d$ $b = a$

ii) reducción in strenght

Esta técnica remplaza operaciones de costo alto con operaciones de menor costo por ejemplo

alto	bajo
x^2	$x \cdot x$

iii) Uso de identidades algebraicas

Esta técnica emplea las propiedades de la asociatividad para eliminar subexpresiones se realiza tambien analizando el DAG por ejemplo

$a = b + c$ $d = b + e + c$ $\Rightarrow$ $a = b + c$ $t = a + e$

5

a)

original

$B \to B\ 0\ R_1$

$B \to B\ 1\ R_2$

$B \to 1\ R_3$

No recursiva izq.

$B \to 1\ R_1\ B'\ R_4$

$B' \to 1\ R_2\ B'_1\ R_5$

$B' \to 0\ R_3\ B'_1\ R_6$

Preserva valor

$B' \to \varepsilon \quad \{B'.s = B'.h\}$

$R_4 \to \varepsilon \quad \{B.val = B'.s\}$

$R_5 \to \varepsilon \quad \{B'.s = B'_1.s\}$

$R_6 \to \varepsilon \quad \{B'.s = B'_1.s\}$

$R_1 \to \varepsilon \quad \{B'.h = 1\}$

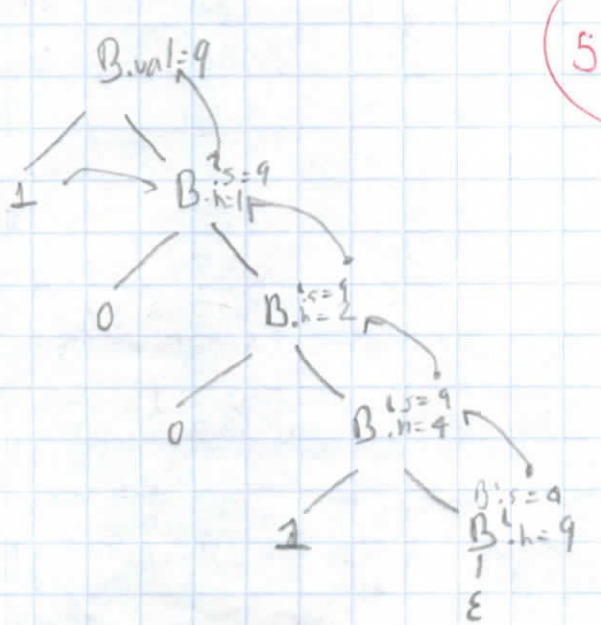
$R_2 \to \varepsilon \quad \{B'_1.h = (B'.h \times 2) + 1\}$

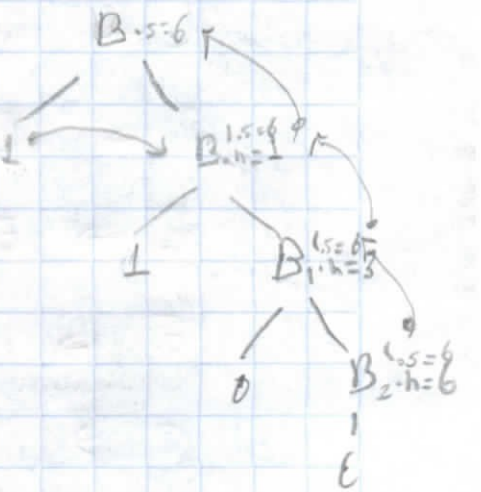
$R_3 \to \varepsilon \quad \{B'_1.h = B'.h \times 2\}$

i) $1001_2 = 9_{10}$

ii) $110_2 = 6_{10}$

5 pts





6- ① Bloque → Declaraciones Sentencias

② Declaraciones → Declaraciones D
③ Sentencias → Sentencias S
④ D → Tipo [C] ID ;
⑤ D → Tipo ID ;
⑥ S → L = E ;
⑦ L → ID [C]
⑧ L → ID
⑨ S → while (R) { Bloque }
⑩ C → ID
⑪ C → num
⑫ R → R < R
⑬ R → E
⑭ E → E + E
⑮ E → C
⑯ Tipo → num

Reglas Semanticas

10 pts

1. -
2. -
3. -
4. ID.tipo = Tipo.val
5. ID.tipo = Tipo.val
6. S.code = L.code || E.code || gen(L.Address '=' E.address ;)
7. t0 = new temp ()
   t0 = Cast to (Type (ID.tipo)) = Cast 4
   L.Address = ID.val [t0]
8. -
9. R.code || Bloque.code ||
10. C.val = id.lexval
11. c.val = num.lexval.

Profesor:

Tercer Examen Parcial
Compiladores

Ejercicio	1	2	3	4	5	6	7	Total
Puntos	5	5	5	5	30	25	25	100
Calificación	5	5	5	5	30	5	20	75

Preguntas

1. (5 puntos) ¿Qué es un sistema de tipos?
2. (5 puntos) ¿Cuál es la diferencia entre un lenguaje que usa tipado estático y uno que utiliza tipado dinámico?
3. (5 puntos) ¿Qué es un bloque básico?
4. (5 puntos) Mencione y explique 3 optimizaciones que pueden ser realizadas sobre un bloque básico.

Ejercicios

5. (30 puntos) La siguiente SDT calcula el valor decimal de una cadena de 0's y 1's interpretada como un entero binario positivo.

$$B \to B_1 0 \{B.val = 2 \times B_1.val\}$$
$$B \to B_1 1 \{B.val = 2 \times B_1.val + 1\}$$
$$B \to 1 \{B.val = 1\}$$

   (a) Reescriba esta SDT de manera que la gramática subyacente no sea recursiva por la izquierda, al mismo tiempo que el valor del atributo $B.val$ de una cadena binaria completa sea preservado.

   (b) Usando su gramática transformada muestre los árboles de parseo anotados para las cadenas binarias `1001` y `110`.

6. (25 puntos) Escriba una SDD que traduzca el siguiente programa a código de tres direcciones. Usando su SDD obtenga el código de tres direcciones equivalente.

```
num[10] a;
num i;
i = 0;
while(i < 10)
{
   a[i] = i;
   i = i + 1;
}
```

(1) Un sistema de tipos es el conjunto de tipos y reglas que siguen estos tipos para construir el comportamiento del lenguaje de programación

5 pts

(2) En un lenguaje con tipado estático, la inferencia de tipo se da en el tiempo de compilación a diferencia del tipado dinámico que se realiza en tiempo de ejecución.

5 pts

(3) Un bloque básico es un conjunto de instrucciones de la representación intermedia en el cual las instrucciones dentro de este son indivisibles en más bloques

5 pts

(4) Un bloque básico se puede optimizar:

- Identificando variables vivas, esto es que las variables que son utilizadas en bloques subsecuentes deben mantenerse en el código pero las muertas pueden desaparecer
- Buscando operadores de menor "costo", es decir operaciones más complejas pueden desarrollarse con operaciones más sencillas
- Utilizando identidades aritméticas, el conjunto de instrucciones puede verse optimizado si se utilizan identidades como son asociatividad, conmutación.

5 pts

(5) a) $B \rightarrow B_1 0$

$B \rightarrow B_1 1$

$B \rightarrow 1$

Eliminando la recursión de la GTC:

$B \rightarrow 1B'$

$B' \rightarrow 0B' \mid 1B' \mid \varepsilon$

Reescribiendo la SDT:

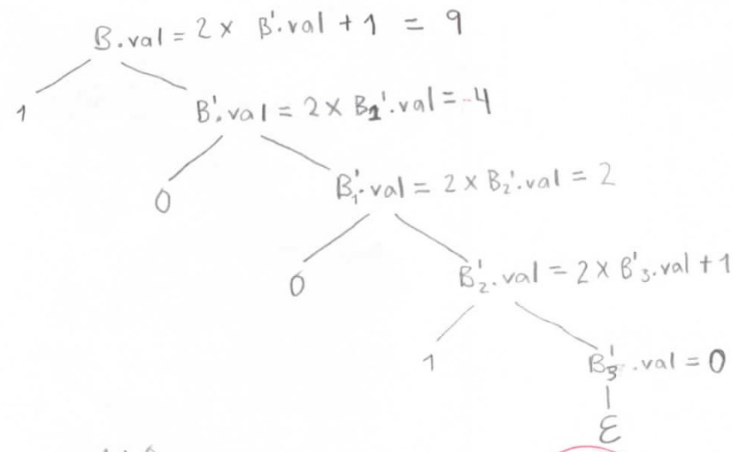
$B \rightarrow 1B' \{B.val = 2 \times B'.val + 1\}$

$B' \rightarrow 0B'_1 \{B'.val = 2 \times B'_1.val\}$

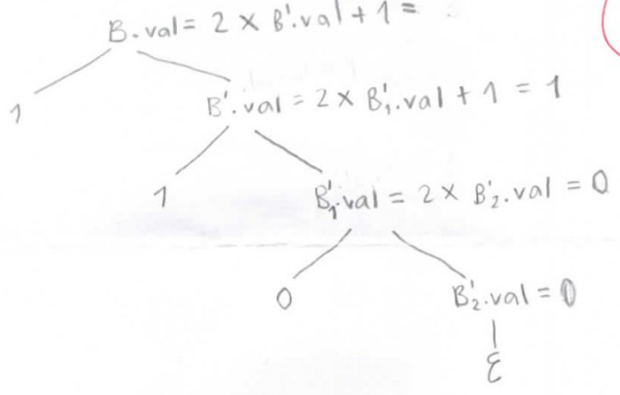
$B' \rightarrow 1B'_1 \{B'.val = 2 \times B'_1.val + 1\}$

$B' \rightarrow \varepsilon \{B'.val = 0\}$

Considerando la cadena 1001



Considerando la cadena 110



30 pts

(6)

D → num D'	
D' → [cte] id; | id;	
S → id assing cte;	S.code = id.lexval || assing.lexval || cte.lexval
S → id[id1] assing id;	S.code = id.lexval || [id1.lexval] || assing.lexval || id.lexval
S → id1 assing id1 op cte;	S.code = t || = || id1.lexval id1.lexval || = || t || op.lexval || cte.lexval
S → while (B) { S1 }	S.code = begin.label || if || (B.code) || goto || B.truelabel goto || B.falselabel || S1.code || goto begin.label
B → id oprel cte	B.code = id.lexval || oprel.lexval || cte.lexval

5 pts

Faltó la traducción
No considerasle la creación de temporales o las direcciones

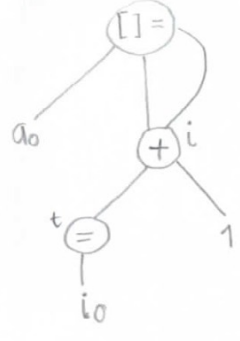
```
i = 0
L1: if ( i < 10) goto L2
    goto L3
L2:
    t = i
    i = t + 1
L3:
```

(7)

```
0
1
2    i = 0
3  L1: if (i < 10) goto L2
4      goto L3
5  L2:
6      t = i
7      i = t + 1
8      goto L1
9  L3:
```

20 pts

DAG (2)



Bloque Básico (1):

```
0
1
2  i = 0
3  L1: if (i < 10) goto L2
4      goto L3
```

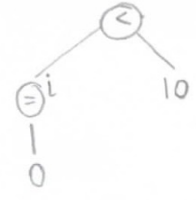
Bloque básico (2):

```
5  L2: a[i] = i
6      t = i
7      i = t + 1
8      goto L1
```

Bloque Básico (3):

```
9  L3:
```

DAG (1):



Profesor:
Mario Martínez Molina

Tercer Examen Parcial
Compiladores
5 de diciembre de 2018

Ejercicio	1	2	3	4	5	6	7	Total
Puntos	5	5	5	5	30	25	25	100
Calificación	5	5	5	5	30	15	25	90

Preguntas

1. (5 puntos) ¿Qué es un sistema de tipos?
2. (5 puntos) ¿Cuál es la diferencia entre un lenguaje que usa tipado estático y uno que utiliza tipado dinámico?
3. (5 puntos) ¿Qué es un bloque básico?
4. (5 puntos) Mencione y explique 3 optimizaciones que pueden ser realizadas sobre un bloque básico.

Ejercicios

5. (30 puntos) La siguiente SDT calcula el valor decimal de una cadena de 0's y 1's interpretada como un entero binario positivo.

$$B \to B_1 0 \{B.val = 2 \times B_1.val\}$$
$$B \to B_1 1 \{B.val = 2 \times B_1.val + 1\}$$
$$B \to 1 \{B.val = 1\}$$

   (a) Reescriba esta SDT de manera que la gramática subyacente no sea recursiva por la izquierda, al mismo tiempo que el valor del atributo $B.val$ de una cadena binaria completa sea preservado.

   (b) Usando su gramática transformada muestre los árboles de parseo anotados para las cadenas binarias `1001` y `110`.

6. (25 puntos) Escriba una SDD que traduzca el siguiente programa a código de tres direcciones. Usando su SDD obtenga el código de tres direcciones equivalente.

```
num[10] a;
num i;
i = 0;
while(i < 10)
{
   a[i] = i;
   i = i + 1;
}
```

1 Es el conjunto conformado por los tipos de datos base (built-in), las reglas de formación de tipos compuestos, así como las formas de interacción entre los tipos y las reglas para determinar los tipos de expresiones.

5 pts

2. tipado estático. La inferencia de tipo para todas las expresiones de un programa fuente se puede realizar en tiempo de compilación.

5 pts

En el tipado dinámico existen algunas expresiones cuyo tipo se puede inferir solo hasta tiempo de ejecución

3. Un conjunto de instrucciones que se asegura se ejecuta siempre. La primer expresión es el único sitio de entrada y la sentencia final es el único sitio de salida.

5 pts

4 Utilización de propiedades aritméticas y algebraicas. Reduce el número de operaciones aplicando identidades matemáticas: $x+0=x$, $x \cdot 1 = 1$, etc.

Eliminación de código muerte. Si se sabe que algunas variables no se utilizan fuera del bloque se pueden eliminar los nodos correspondientes en el DAG

Código duplicado. Se evita calcular de nuevo una expresión ya calculada

5 pts

5

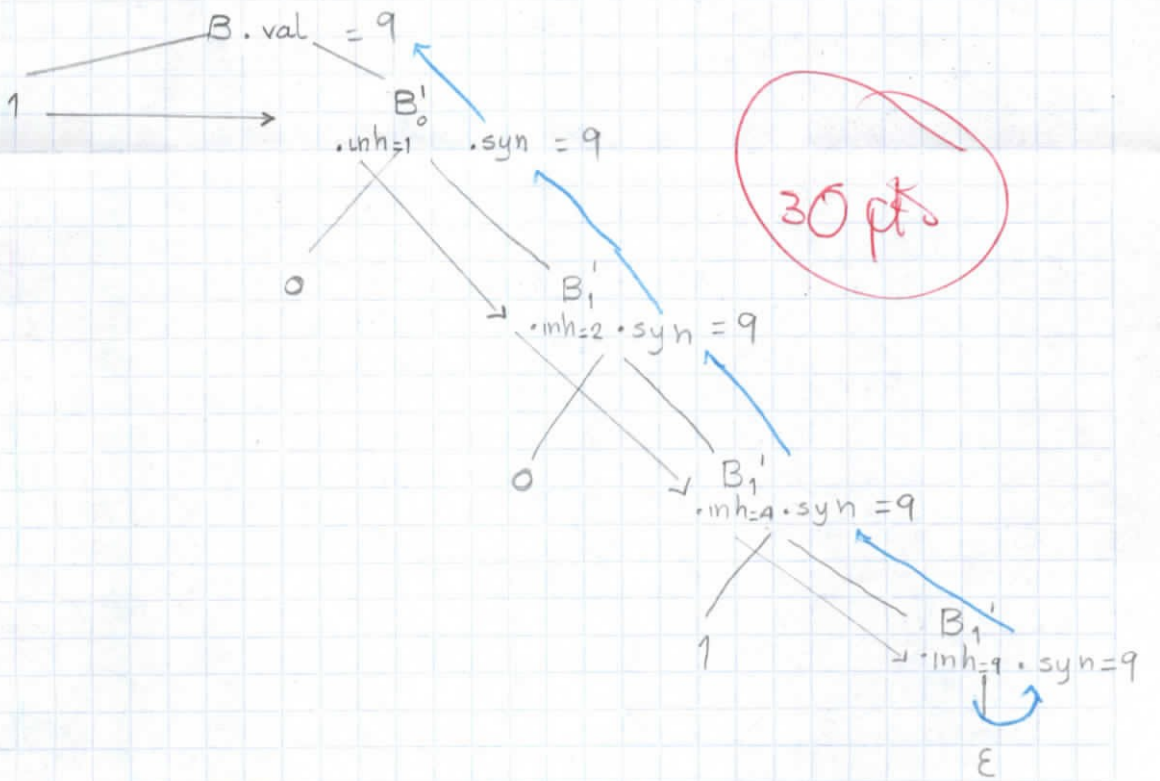
5.

(a)

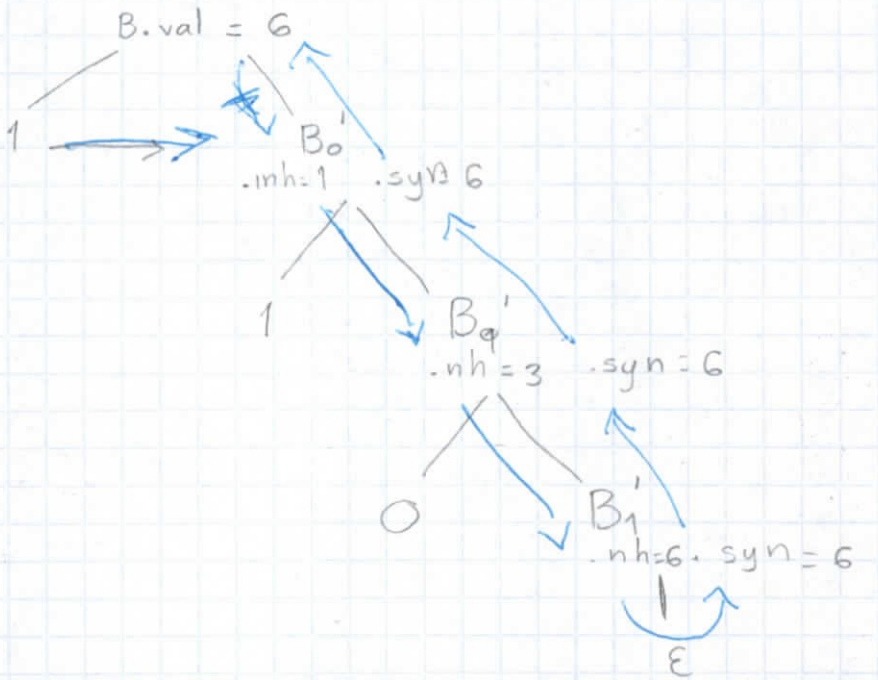
$B \rightarrow 1 \quad \{ B'.inh = 1 \}$
$\quad\quad B' \quad \{ B.val = B'.syn \}$

$B'_0 \rightarrow 0 \quad \{ B'_1.inh = 2 * B'_0.inh \}$
$\quad\quad B'_1 \quad \{ B'_0.syn = B'_1.syn \}$
$\quad | \quad 1 \quad \{ B'_1.inh = 2 * B'_0.inh + 1 \}$
$\quad\quad B'_1 \quad \{ B'_0.syn = B'_1.syn \}$
$\quad | \quad \varepsilon \quad \{ B'_0.syn = B'_0.inh \}$

(b)



30 pts


B.val = 6
1
B_0'
.inh = 1
.syn = 6
1
B_0'
.inh = 3
.syn = 6
0
B_1'
.inh = 6
.syn = 6
1
ε

6

Reglas semánticas

1 Block → Declarations Statemets

2 Declarations → Declaration Declarations'
3 | ε

4 Declaration → num Specifier

5 Specifier → [Expr] Specifier
6 | id ;

7 Statemets$_0$ → Loc = Expr ;
8 | while (Bexpr) { Statements$_1$ }

9 Loc → id Loc'
10 Loc' → [Expr]
11 | ε

12 Bexpr → Term$_1$ < Term$_2$
13 Expr | Term$_1$ + Term$_2$

14 Term → number
15 | id

15. Term.addr = id.lexval Term.code = ''

14. Term.addr = number.lexval Term.code = ''

13 Expr.addr = new Temp() Expr.code = $Term_1$.code ||
$Term_2$.code
gen($Term_1$.addr '+' $Term_2$.addr)

12 Bexpr.code = $Term_1$.code ||
$Term_2$.code ||
gen('if' $Term_1$.addr '<' $Term_2$.addr
goto Bexpr.True
goto Bexpr.False)

9 Loc.base = id.lexval Loc'.inh = Loc.base
Loc.type = Type(Loc.base)
Loc.addr = new Temp()
Loc.code = gen(Loc.addr '=' Loc'.syn)

10
Loc'.base = Loc'.inh Loc'_1.inh = Loc_1.inh
Loc.type = Type(Loc.base)
Loc.addr = new Temp() t = new Temp()
Loc.code = Expr.code ||
gen('t' '=' '4' '*' Expr.addr)
gen(Loc.addr '=' 't')

11 Loc'.syn = Loc'.inh

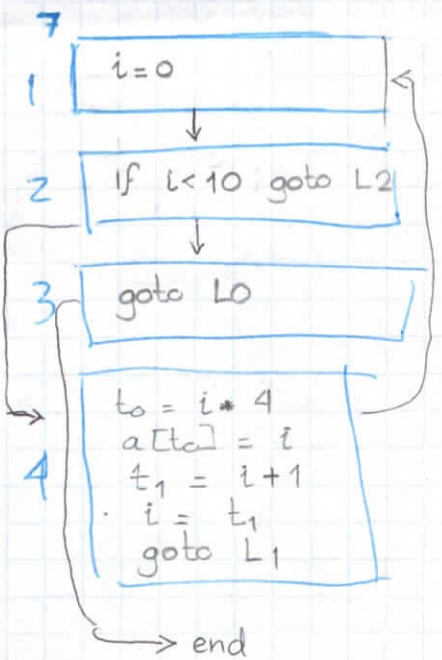
7 gen(Labase[Loc.addr] '=' Expr.addr)

8 St.begin = new Label()
B.True = new Label()
B.False = S.next
St_1.next = St.begin
St.code = Label(St.begin) ||
Bexpr.code ||
St_1.code ||
gen(goto St.begin)

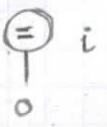
15 pts

1. S.next = new Label
B.code = Statements.code(
label(S.next))

```
    i = 0
L1: if i < 10
        goto L2
        goto Lo
L2: to = i * 4
    a[to] = i
    t1 = i+1
    i = t1
    goto L1
Lo:
```

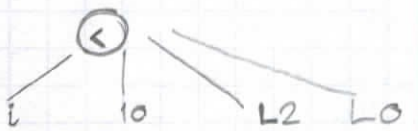


1.



25 pts

2



3