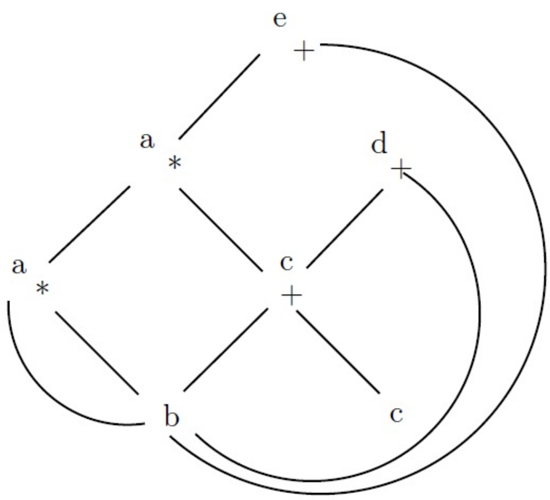
German University in Cairo
Department of Computer Science
Assoc. Prof. Haythem O. Ismail

CSEN 1003 Compiler, Spring Term 2019
Practice Assignment 11

Exercise 11-1

The three-address code is a linearized version of the DAG. Given the following Directed Acyclic Graph, generate the corresponding three-address code.



Solution:

```
c = b + c
a = b * b
d = c + b
a = a * c
e = a + b
```

Exercise 11-2

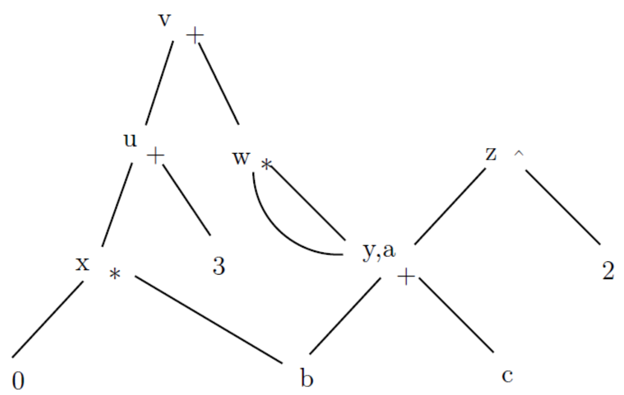
Consider the following block of three-address code, in which all variables are integers, and denotes exponentiation.

```
a=b+c
z=a^2
x=0*b
y=b+c
```

```
w=y*y
u=x+3
v=u+w
```

Derive the directed-acyclic graph for this basic block

Solution:



Exercise 11-3

a) Extend the SDD in Lecture 10, slide 6 by adding a translation rule for the following production:

$E \quad \rightarrow \quad E_1 * E_2$

Solution:

Production	Semantic Rule
$E \quad \rightarrow \quad E_1 * E_2$	$E.addr = \textbf{new}\ Temp();$ $E.code = E_1.code$ • $E_1.code;$ • gen(E.addr '=' $E_1.addr'*'E_2.addr);$

b) Convert the extended SDD to an SDT.

Solution:

$S \rightarrow \textbf{id} = E\ ;$ $\quad \{\ gen(\ top.get(\textbf{id}.lexeme)\ '='\ E.addr);\ \}$

$E \rightarrow E_1 + E_2$ $\quad \{\ E.addr = \textbf{new}\ Temp\,();$
$gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr);\ \}$

$\mid\ -\ E_1$ $\quad \{\ E.addr = \textbf{new}\ Temp\,();$
$gen(E.addr\ '='\ '\textbf{minus}'\ E_1.addr);\ \}$

$\mid\ (\ E_1\ )$ $\quad \{\ E.addr = E_1.addr;\ \}$

$\mid\ \textbf{id}$ $\quad \{\ E.addr = top.get(\textbf{id}.lexeme);\ \}$

c) Use the extended SDT to translate the following assignments:

1. `a = b + -c.`

   Solution:

   ```
   t1 = minus  c
   t2 = b + t1
   a = t2
   ```

2. `a = b + (c * d).`

   Solution:

   ```
   t1 = c * d
   t2 = b + t1
   a = t2
   ```

The corresponding 3-Address code translation is generated is by constructing the parse tree for the string and evaluating the attributes of the node variables whenever possible. In the above grammar, a bottom up order evaluation is suitable since the grammar is S-attributed.

Exercise 11-4

a) Convert the SDD in lecture 10, slide 13 to an SDT.

Solution:

$$
\begin{array}{lll}
S & \rightarrow \ \mathbf{id} = E\ ; & \{\ gen(\ top.get(\mathbf{id}.lexeme)\ '='\ E.addr);\ \} \\
 & \mid\ L = E\ ; & \{\ gen(L.array.base\ '['\ L.addr\ ']'\ '='\ E.addr);\ \} \\
E & \rightarrow\ E_1 + E_2 & \{\ E.addr = \mathbf{new}\ \mathit{Temp}\,(); \\
 & & \quad gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr);\ \} \\
 & \mid\ \mathbf{id} & \{\ E.addr = top.get(\mathbf{id}.lexeme);\ \} \\
 & \mid\ L & \{\ E.addr = \mathbf{new}\ \mathit{Temp}\,(); \\
 & & \quad gen(E.addr\ '='\ L.array.base\ '['\ L.addr\ ']');\ \} \\
L & \rightarrow\ \mathbf{id}\ [\ E\ ] & \{\ L.array = top.get(\mathbf{id}.lexeme); \\
 & & \quad L.type = L.array.type.elem; \\
 & & \quad L.addr = \mathbf{new}\ \mathit{Temp}\,(); \\
 & & \quad gen(L.addr\ '='\ E.addr\ '*'\ L.type.width);\ \} \\
 & \mid\ L_1\ [\ E\ ] & \{\ L.array = L_1.array; \\
 & & \quad L.type = L_1.type.elem; \\
 & & \quad t = \mathbf{new}\ \mathit{Temp}\,(); \\
 & & \quad L.addr = \mathbf{new}\ \mathit{Temp}\,(); \\
 & & \quad gen(t\ '='\ E.addr\ '*'\ L.type.width); \\
 & & \quad gen(L.addr\ '='\ L_1.addr\ '+'\ t);\ \}
\end{array}
$$

b) Assume that a is a 2x3 array of integers, b is a 4x5 array of integers, i and j are integers and the width of an integer is 4. Use the SDT from part a to translate the following assignments:

1. `x = a[i] + b[j].`

   Solution:

   ```
   t1 = i * 12
   t2 = a[t1]
   t3 = j * 20
   t4 = b[t3]
   t5 = t2 + t4
   x = t5
   ```

2. `x = a[i][j] + b[i][j].`

   Solution:

   ```
   t1 = i * 12
   t2 = j * 4
   t3 = t1 + t2
   t4 = a[t3]
   t5 = i * 20
   t6 = j * 4
   t7 = t1 + t2
   t8 = b[t3]
   t9 = t4 + t8
   x = t9
   ```

The corresponding 3-Address code translation is generated is by constructing the parse tree for the string and evaluating the attributes of the node variables whenever possible. In the above grammar, a bottom up order evaluation is suitable since the grammar is S-attributed.

Exercise 11-5

An integer array $A[i, j]$ has index i ranging from 0 to 10 and index j ranging from 0 to 20. Integers take 4 bytes each. Suppose array A is sorted starting at byte 0. Find the location of the following:

a) $A[4, 5]$

   Solution:

   $A[4, 5] = 0 + 4 * 21 * 4 + 5 * 4$

b) $A[10, 8]$

   Solution:

   $A[10, 8] = 0 + 10 * 21 * 4 + 8 * 4$

c) $A[3, 17]$

   Solution:

   $A[3, 17] = 0 + 3 * 21 * 4 + 17 * 4$

Exercise 11-6

Repeat Exercise 11-3 if A is sorted in column-major order.

a) $A[4,5]$

   Solution:

   $A[4,5] = 0 + 4 * 4 + 5 * 11 * 4$

b) $A[10,8]$

   Solution:

   $A[10,8] = 0 + 10 * 4 + 8 * 11 * 4$

c) $A[3,17]$

   Solution:

   $A[3,17] = 0 + 3 * 4 + 17 * 11 * 4$

Exercise 11-7

A real array $A[i,j,k]$ has index i ranging from 0 to 4 and index j ranging from 0 to 4, and index k ranging from 0 to 10. Reals take 8 bytes each. Suppose array A is stored starting at byte 0. Find the location of the following:

a) $A[3,4,5]$

   Solution:

   $A[3,4,5] = 0 + 3 * 5 * 11 * 8 + 4 * 11 * 8 + 5 * 8$

b) $A[1,2,7]$

   Solution:

   $A[1,2,7] = 0 + 1 * 5 * 11 * 8 + 2 * 11 * 8 + 7 * 8$

c) $A[4,3,9]$

   Solution:

   $A[4,3,9] = 0 + 4 * 5 * 11 * 8 + 3 * 11 * 8 + 9 * 8$

Exercise 11-8

Repeat Exercise 11-5 if A is sorted in column-major order.

a) $A[3,4,5]$

   Solution:

   $A[3,4,5] = 0 + 3 * 8 + 4 * 5 * 8 + 5 * 5 * 5 * 8$

b) $A[1,2,7]$

Solution:

$A[1,2,7] = 0 + 1 * 8 + 2 * 5 * 8 + 7 * 5 * 5 * 8$

c) $A[4,3,9]$

Solution:

$A[4,3,9] = 0 + 4 * 8 + 4 * 5 * 8 + 9 * 5 * 5 * 8$

Exercise 11-9

The following is an SDD for programs with simple statements and Boolean expressions.

$$
\begin{array}{lllll}
P & \longrightarrow & S & & S.next = newlabel() \\
 & & & P.code & = S.code \circ label(S.next) \\
 & & & & \\
S & \longrightarrow & \mathbf{id}_1 {=} \mathbf{id}_2 + \mathbf{id}_3 & S.code & = gen(\mathbf{id}_1.addr \; '=' \; \mathbf{id}_2.addr \; '+' \; \mathbf{id}_3.addr) \\
 & & & & \\
S & \longrightarrow & \textbf{while } (B) \; S_1 & & B.true = newlabel(); B.false = S.next \\
 & & & & S_1.next = newlabel() \\
 & & & S.code & = label(S1.next) \;\circ\; B.code \\
 & & & & \circ \; label(B.true) \;\circ\; S_1.code \\
 & & & & \circ \; gen('\texttt{goto}' \; S_1.next) \\
 & & & & \\
B & \longrightarrow & B_1 \;\texttt{\&\&}\; B_2 & & B_1.true = newlabel(); B_1.false = B.false; \\
 & & & & B_2.true = B.true; B_2.false = B.false; \\
 & & & B.code & = B_1.code \circ label(B_1.true) \circ B_2.code \\
 & & & & \\
B & \longrightarrow & \mathbf{id}_1 == \mathbf{id}_2 & B.code & = gen('\texttt{if}' \; \mathbf{id}_1.addr \; '==' \; \mathbf{id}_2.addr \; '\texttt{goto}' \; B.true) \\
 & & & & \circ \;\; gen('\texttt{goto}' \; B.false)
\end{array}
$$

Give the value of `P.code` as a result of parsing the string

```
while (x==y && z==u) while (x == u) x = z + y
```

Assume that generated labels are in the form `Li`, where i is an integer indicating the order in which the labels are generated; thus, the first label is `L1`, the second `L2`, and so on. (Assume top-down parsing. That is, labels generated closer to the root of the parse tree are generated earlier.)

Solution:

```
L3: if x == y goto L4
    goto L1
L4: if z == u goto L2
    goto L1
L2: L6: if x == u goto L5
        goto L3
L5: x = z + y
    goto L6
    goto L3
L1:
```