

Compiler Homework 3

1 (a) Use attributed grammar to check the semantic rules given above. In other words, you should add attribute rules to the given grammar. Only inherited and synthesized attributes are allowed, no global attributes.

$P \rightarrow S; P$	$P.level = \max(S.level, P.level)$ $P.IDs = S.IDs \cup P.IDs$
$P \rightarrow S;$	$P.IDs = S.IDs$ $P.level = S.level$
$S \rightarrow A$	$S.IDs = A.IDs$ $S.level = 0$
$S \rightarrow R$	$S.IDs = R.IDs$ $S.level = 0$
$S \rightarrow F$	$S.IDs = F.IDs$ $S.level = F.level$
$A \rightarrow id := E$	$IDs = \{id\}$ $Id.val = E.val$ If $Id.type \neq E.type$, error
$E \rightarrow E1 + id$	$E.type = E1.type$ $E.val = E1.val + id.val$
$E \rightarrow id$	$E.val = id.val$
$E \rightarrow O$	$E.type = O.type$ $E.val = O.val$
$E \rightarrow (M)$	$E.list = M.list$
$R \rightarrow \text{print id}$	$\text{Print}(id.val)$
$F \rightarrow \text{foreach id in L do } \{ P \}$	$F.level = P.level + 1;$ If $F.level > 2$, error; If $\{id\}$ in $P.IDs$, error; $F.IDs = \{id\} \cup P.IDs$
$L \rightarrow (M)$	$L.list = M.list$
$L \rightarrow id$	$L.val = id.val$ $L.type = id.type$
$M \rightarrow M1 O$	If $M1.type = \text{void}$, $M.type = O.type$; If $M1.type \neq O.type$, error; $M.list = \text{append}(M1.list, O.val)$
$M \rightarrow \epsilon$	$M.type = \text{void}$
$O \rightarrow \text{string}$	$O.type = \text{string}$ $O.val = \text{string.lexval}$
$O \rightarrow \text{integer}$	$O.type = \text{integer}$ $O.val = \text{integer.lexval}$

1 (b) If your attributed grammar can be evaluated during parsing without an additional evaluation phase, then briefly discuss how it is possible. If your grammar cannot be evaluated at the parsing phase, then use the techniques we have introduced to modify your attributed grammar or the original grammar so that the semantic analysis can be done with parsing.

Because the attribute rules above are all synthesized rules, they can be evaluated during parsing without an additional evolution.

2. Consider a given input statement: "member [w, x, y, z]:= 103". Follow the attribute grammar above and generate the three address code.

```
t1 = w
t1 = t1 * arraySize[1]
t1 = t1 + x;
t1 = t1 * arraySize[2]
t1 = t1 + y;
t1 = t1 * arraySize[3]
t1 = t1 + z;
t1 = t1 * elementWidth;
member[t1]=103
```

3. As you can see, it is not possible for the LL parsing algorithm to evaluate the attribute grammar (evaluate the input expression). Discuss why not.

LL parsing is a top-down parsing. Generally the "top" are non-terminals and contains no information from the input. When the top non-terminals are expanded to the bottom, the relations between the siblings are no longer kept anywhere. So the information extracted from input cannot flow to the places that need it.

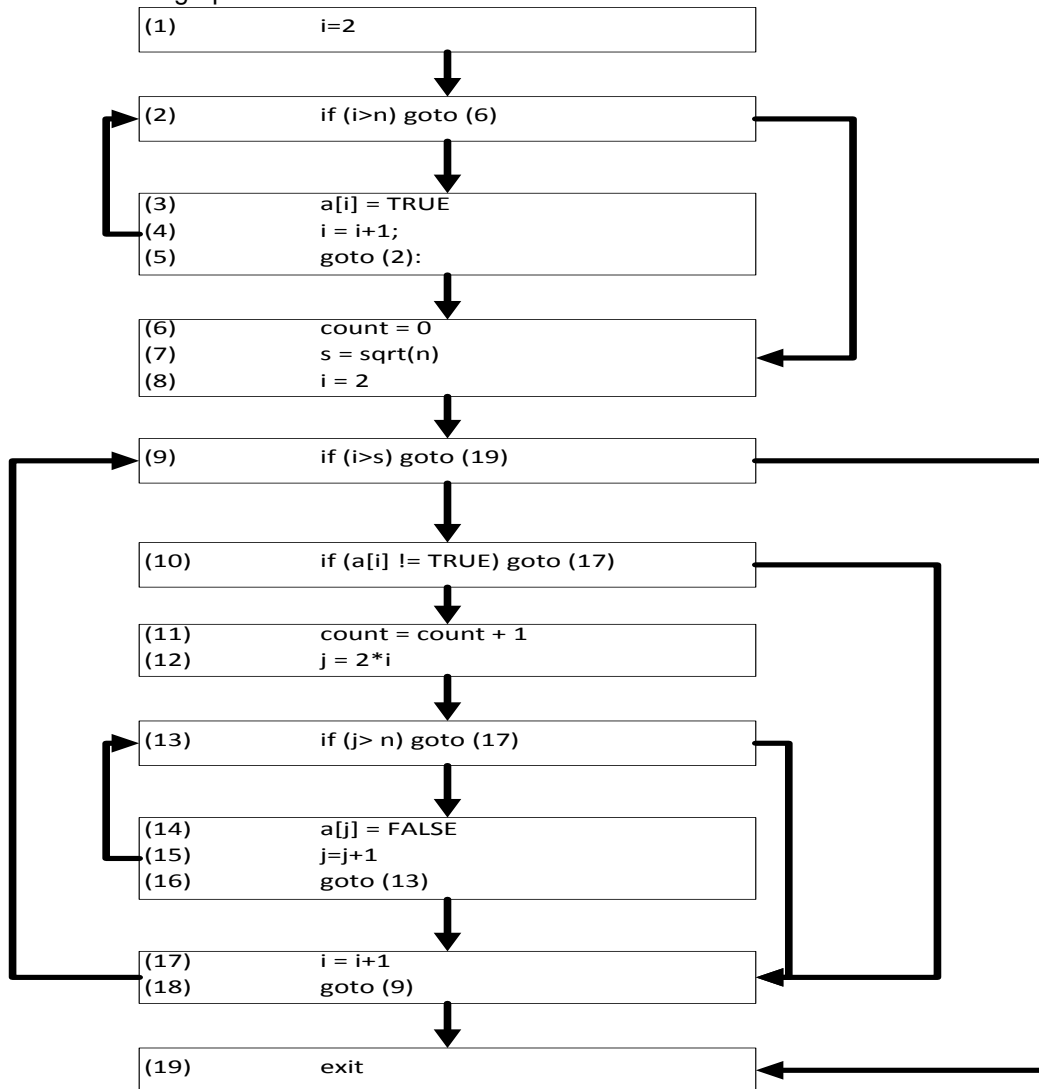
4(a) Translate the program into three address code as defined in Section 6.2, dragon book.

```
(1)   i=2
(2)   if (i>n) goto (6)
(3)   a[i] = TRUE
(4)   i = i+1;
(5)   goto (2):
(6)   count = 0
(7)   s = sqrt(n)
(8)   i = 2
(9)   if (i>s) goto (19)
(10)  if (a[i] != TRUE) goto (17)
(11)  count = count + 1
(12)  j = 2*i
(13)  if (j> n) goto (17)
(14)  a[j] = FALSE
(15)  j=j+1
(16)  goto (13)
(17)  i = i+1
(18)  goto (9)
(19)  exit
```

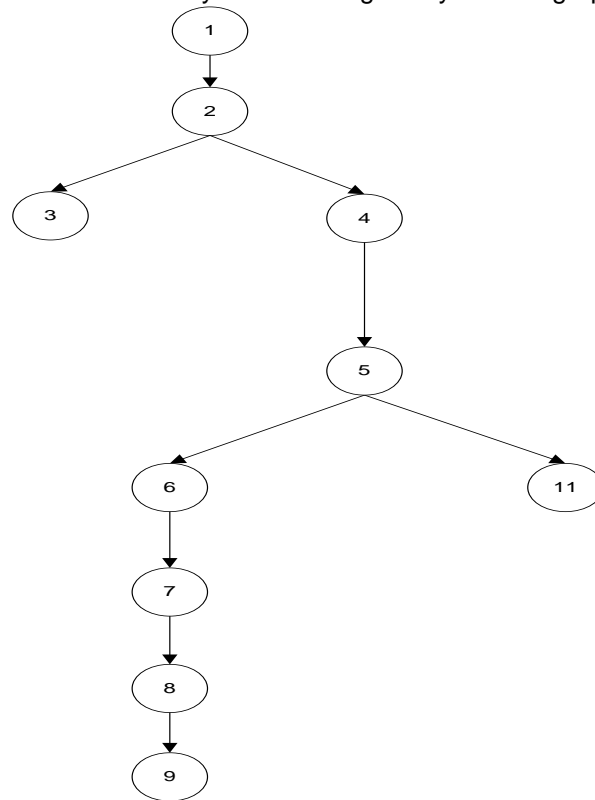
4(b) Identify all basic blocks in your three address code.

(1)	i=2
(2)	if (i>n) goto (6)
(3)	a[i] = TRUE
(4)	i = i+1;
(5)	goto (2):
(6)	count = 0
(7)	s = sqrt(n)
(8)	i = 2
(9)	if (i>s) goto (19)
(10)	if (a[i] != TRUE) goto (17)
(11)	count = count + 1
(12)	j = 2*i
(13)	if (j> n) goto (17)
(14)	a[j] = FALSE
(15)	j=j+1
(16)	goto (13)
(17)	i = i+1
(18)	goto (9)
(19)	exit

4(c) Build the flow graph for the three address code.



4(d) Build the dominator tree and identify the back edges in your flow graph in (c).



Back edge (3, 2) (10, 5) (9,8)

4(e) Find the entry node and the set of nodes in the natural loop associated with each back edge identified in (d).

- (3,2), entry node is 2, the set of nodes is {2,3}
- (10,5), entry node is 5, the set of nodes is {5,6,7,8,9,10}
- (9,8), entry node is 8, the set of nodes is {8,9}