BRAC UNIVERSITY

Department of Computer Science and Engineering

Examination: Final Examination | Semester: Spring 2025

Duration: 1 Hour 20 Minutes | **Set - A** | Full Marks: 40

CSE 420: Compiler Design

Figures in the right margin indicate marks.

Answer all the questions

COs	Questions	Marks
CO4	1. What are the attributes we store in the symbol table for each variable and why do we need to store this information for code generation, explain with examples.	10

CO5	2. Convert the following statements into Three Address Code and show the Quadruple representation of it.
	a = c - b * t * (- b);
	getNeuronBias((c + d) * (- c), b * (x / t));
		5

CO5	3. Consider the following **SDT.** Draw the Annotated Parse Tree and evaluate the tree for the string: ***a[i+j] = b[x][y];*** (Annotate the attributes "addr", "type", "idx", "off", also show the value of "tScale" when it is being updated at certain nodes and show the generated three address code) **Note: Here "a" is 1 dimensional integer array with size 10 and "b" is 2 dimensional integer array with size 5 x 7. The type of a is array(10, integer). Its width w is 40, assuming that the width of an integer is 4. The type of b is array(5, array(7, integer)). Its width w is 140, assuming that the width of an integer is 4. The new Temp() creates an instance of compiler generated temporary variables which are 't' with a subscript 1, 2, 3 and so on. For example, the first time an instance is created it will create t1, next time it is called it will create t2 and so on. The gen method generates the three address codes.**	10

	SDT
	
	S $\rightarrow$ id = E ; { gen(top.get(id.lexeme) " = " E.addr); }
	
	| L = E ; { gen(L.base "[" L.off "] = " E.addr); }
	
	E $\rightarrow$ E_1 + E_2 { E.addr = new Temp(); gen(E.addr " = " E_1.addr " + " E_2.addr); }
	
	| id { E.addr = top.get(id.lexeme); }
	
	| L { E.addr = new Temp(); gen(E.addr " = " L.base "[" L.off "]"); }
	
	L $\rightarrow$ id [E] { L.array = top.get(id.lexeme); L.type = L.array.type.elem;
	
	L.base = L.array.base; L.idx = E.addr;
	
	L.off = new Temp(); gen(L.off " = " L.idx " * " L.type.width);}
	
	| L_1 [E] { L.array = L_1.array; L.type = L_1.type.elem; L.base = L_1.base;
	
	L.idx = E.addr; tScale = new Temp()
	
	gen(tScale " = " L.idx " * " L.type.width);
	
	L.off = new Temp(); gen(L.off " = " L_1.off " + " tScale); }	

CO5	4. Consider the following **SDT.** Draw the **Annotated Parse Tree** and **Evaluate** it for the String: ***while (x < y) z = u + v - x ; y = z * u ;*** **Note: The newlabel() and newtemp() function details have been given to you** *(5 points for parse tree + 5 points for proper annotation of labels and temp variables for expressions in the parse tree + 5 points for final three address code)*	15

SDT:

```
int newlabel() {
    return "L" || p;
}
int newtemp() {
    return 't' || p;
}

P -> {p = 0; S.next = newlabel();} S {P.code = S.code || label(S.next)}
S -> {p++;} Assign {S.code = Assign.code;}
Assign -> id = E; {Assign.code = E.code || gen(id.lexeme = E.addr);}
S -> if ( {p++; B.true = newlabel(); B.false = $S_1$.next = S.next;}
          B ) {p++}
          $S_1$ {S.code = B.code || label(B.true) || $S_1$.code;}
S -> if ( {p++; B.true = newlabel() || "_T"; B.false = newlabel() || "_F";}
          B ) {p++; $S_1$.next = S.next} $S_1$ else {p++; $S_2$.next = S.next}
          $S_2$ {S.code = B.code || label(B.true) || $S_1$.code || gen('goto' S.next) ||
                    label(B.false) ||  $S_2$.code;}
S -> while ( {p++;  begin = newlabel(); B.true = newlabel() || "_T"; B.false = S.next;}
          B ) {p++; $S_1$.next = begin;}
          $S_1$ {S.code = label(begin) || B.code || label(B.true) || S1.code || gen('goto'
                    begin);}
S -> {p++; $S_1$.next = newlabel();}
          $S_1$ {p++; $S_2$.next = S.next;}
          $S_2$ {S.code = S1.code || label(S1.next) || S2.code;}
B -> {p++;}
          $E_1$ rel {p++;}
          $E_2$ {B.code = $E_1$.code || $E_2$.code
                    || gen('if' $E_1$.addr  rel.op $E_2$.addr 'goto' B.true); || gen('goto' B.false)}
E -> {p++;}
          $E_1$ arith {p++;}
          $E_2$ {E.addr = newtemp(); E.code = $E_1$.code || $E_2$.code || gen(E.addr = $E_1$.addr
                    arith.op $E_2$.addr);}
E -> id {E.addr = id.lexeme; E.code = "";}
E -> num {E.addr = num.value; E.code = "";}
```