CS31003 Compilers
Class Test 2
30-Oct-2025, 6:00pm–7:00pm
Maximum Marks: 50

Roll No: _____

Name: _____

[*Write your answers in the question paper itself. Be brief and precise.*]

1. Let `P` and `Q` be two-dimensional integer arrays of dimension `3 x 4`, declared as follows.

```
int P[3][4], Q[3][4];
```

(a) Write the type expressions for the arrays `P` and `Q`. **[3]**

```
array(3, array(4, integer))
```

(b) Let `i`, `j`, `k`, and `z` all denote integer variables. Consider the following assignment statement.

```
z = 1 + P [ Q [ i + 2 ] [ j * 3 ] ] [ k - 4 ] ;
```

Translate the above assignment statement to its corresponding Three-Address Intermediate Code. Assume that the width of an integer is 4. **[10]**

```
t1 = i + 2
t2 = t1 * 16
t3 = j * 3
t4 = t3 * 4
t5 = t2 + t4
t6 = Q [ t5 ]
t7 = t6 * 16
t8 = k - 4
t9 = t8 * 4
t10 = t7 + t9
t11 = P [ t10 ]
t12 = 1 + t11
z = t12
```

2. In this question, you are supposed to write the semantic actions for a set of productions for translating `do…while` statements in a program. Assume that the program starts with the non-terminal `P` of the grammar, and `B` (capturing the Boolean expressions of the grammar) and `S` (capturing statements of the grammar) have a synthesized attribute `code`, which gives the translation into three-address instructions upon translations of `B.code` and `S.code` as strings. The labels of `B.code` and `S.code` may be managed using inherited attributes. To a Boolean expression `B`, associate two labels: `B.true` (the label to which control flows if `B` evaluates to `true`) and `B.false` (the label to which control flows if `B` evaluates to `false`). Furthermore, to each statement `S`, associate an inherited attribute `S.next` denoting a label for the instruction immediately after `S`. For uniformity, use the following functions in your semantic rules.

- `newlabel()` to create a new label,
- `label(L)` to attach the label "`L:`" to the next three-address instruction to be generated,
- `newtemp()` to create a new temporary, and
- `gen()` to generate the three-address code string.

Here, `do`, `while`, {, }, (,), +, –, &&, <, > and `;` are lexical tokens (including keywords) carrying the conventional meanings as in the C language. Moreover, `id` and `num` respectively denote identifier/variable and number whose `lexval` is obtained from the lexical analyzer.

(a) Clearly write the semantic rules (in the right column) corresponding to each production (shown on the left column), in the table below. Do not use any fall-through code optimization in this part. **[16]**

Productions	**Semantic Rules**
`P ⟶ S`	`S.next = newlabel()` `P.code = S.code + label(S.next)` [+ denotes concatenation of strings]

`S ⟶ do { S1 } while (B) ;`
[3 marks]
	`REPEAT = newlabel()`
	`S1.next = newlabel()`
	`B.true = REPEAT`
	`B.false = S.next`
	`S.code = label(REPEAT) + S1.code + label(S1.next) + B.code`

`S ⟶ S1 S2`
[2 marks]
	`S1.next = newlabel()`
	`S2.next = S.next`
	`S.code = S1.code + label(S1.next) + S2.code`

`S ⟶ id ++ ;` **[2 marks]**	`T = newtemp()` `S.code = gen( T '=' id.lexval '+ 1' ) + gen( id.lexval '=' T )`
`S ⟶ id -- ;` **[2 marks]**	`T = newtemp()` `S.code = gen( T '=' id.lexval '- 1' ) + gen( id.lexval '=' T )`

Productions	Semantic Rules
B ⟶ B1 && B2 **[3 marks]**	B1.true = newlabel() B1.false = B.false B2.true = B.true B2.false = B.false B.code = B1.code + label(B1.true) + B2.code
B ⟶ id1 < id2 **[2 marks]**	B.code = gen('if' id1.lexval '<' id2.lexval 'goto' B.true) + gen('goto' B.false)
B ⟶ id > num **[2 marks]**	B.code = gen('if' id.lexval '>' num.lexval 'goto' B.true) + gen('goto' B.false)

(b) Now, translate the following code fragment into the three-address code following the semantic rules that you presented in part (a) above: `do { i ++ ; j -- ; } while ( j > 0 && i < j ) ;` **[6]**

```
REPEAT: t1 = i + 1
        i = t1

    L4: t2 = j - 1
        j = t2

    L3: if j > 0 goto L2
        goto L1

    L2: if i < j goto REPEAT
        goto L1

    L1:
```

(c) Eliminate any redundant `goto` instructions that you generated in the three-address code of part (b) when using the semantics rules from part (a). Only rewrite the semantic rules for the corresponding productions handing the *"falling through"* optimizations that eliminate redundant `goto` instructions. No need to write the semantic rules for the other productions where there is no change. **[10]**

Productions	Semantic Rules
S ⟶ do { S1 } while (B) ; **[3 marks]**	REPEAT = newlabel() S1.next = newlabel() B.true = REPEAT B.false = fall S.code = label(REPEAT) + S1.code + label(S1.next) + B.code

B ⟶ B1 && B2 **[3 marks]**	B1.true = fall B1.false = **if** B.false ≠ fall **then** B.false **else** newlabel() B2.true = B.true B2.false = B.false B.code = **if** B.false ≠ fall **then** B1.code + B2.code **else** B1.code + B2.code + label(B1.false)
B ⟶ id1 < id2 **[2 marks]**	B.code = **if** B.true ≠ fall **and** B.false ≠ fall **then** gen('if' id1.lexval '<' id2.lexval 'goto' B.true) + gen('goto' B.false) **else if** B.true ≠ fall **then** gen('if' id1.lexval '<' id2.lexval 'goto' B.true) **else if** B.false ≠ fall **then** gen('ifFalse' id1.lexval '<' id2.lexval 'goto' B.false) **else** ' '
B ⟶ id > num **[2 marks]**	B.code = **if** B.true ≠ fall **and** B.false ≠ fall **then** gen('if' id.lexval '>' num.lexval 'goto' B.true) + gen('goto' B.false) **else if** B.true ≠ fall **then** gen('if' id.lexval '>' num.lexval 'goto' B.true) **else if** B.false ≠ fall **then** gen('ifFalse' id.lexval '>' num.lexval 'goto' B.false) **else** ' '

(d) Again, translate the code fragment given in part (b) into the three-address code with redundant `goto` instructions eliminated following the changed semantic rules that you formulated in part (c). **[5]**

```
REPEAT: t1 = i + 1
        i = t1

    L4: t2 = j - 1
        j = t2

    L3: ifFalse j > 0 goto L1

        if i < j goto REPEAT

    L1:
```