COMP36512 – May/June 2019

MODEL ANSWERS to exam questions
(3 pages)

QUESTION 1
Clearly: 1-b; 2-h; 3-f; 4-g; 5-c; 6-a; 7-i; 8-d; 9-e.
1 mark for each correct answer.
(9 marks)

QUESTION 2
(i) this would require a modification of the lexical analyser – nothing else.
(ii) this would require some modifications of the low-level code optimiser – nothing to do with the front-end.
(**4 marks**, 2 marks each)

QUESTION 3
This transformation is a combination of loop skewing (add i to the innermost loop of the original code) and loop interchange (new loop bounds have to be calculated). This is a typical wavefront computation. In the original version, none of the loops is parallelizable because of the loop-carried dependences. In the transformed version, different iterations of the i loop can be executed in parallel, as the optimization groups the dependences for different iterations of the j loop. Students have seen a range of transformations and/or optimisations and their impact was discussed in the lectures (including the impact on code parallelization), but the specific example (parallelization of a wavefront computation) was not shown to them.
(5 marks)

QUESTION 4
This language generates strings containing any number of a or b in any order, with the final symbol being always a. For example: a, aa, ba, aaa, aba, baa, bba, etc…
(2 marks)

QUESTION 5
Clearly, all such integers should end with one of 00, 25, 50, 75. Then, the regular expression needs to generate all 5-digit integers between 40000 (the lowest multiple of 25 which is greater than 39980) and 99999 that obey this rule. Hence:
RE → (4|5|6|7|8|9) (digit) (digit) (00 | 25 | 50 | 75)
(3 marks)

QUESTION 6

i) The relevant steps taken are:

Stack	Input	Action taken
$ 0	(( )) ( )	Shift 3
$ 0 (3	( )) ( )	Shift 6
$ 0 (3 (6	)) ( )	Shift 10
$ 0 (3 (6) 10	) ( )	Reduce 5
$ 0 (3 P 5	) ( )	Shift 8
$ 0 (3 P 5) 8	( )	Reduce 4
$ 0 P 2	( )	Reduce 3
$ 0 L 1	( )	Shift 3
$ 0 L 1 (3	)	Shift 7
$ 0 L 1 (3) 7	eof	Reduce 5
$ 0 L 1 P 4	eof	Reduce 2
$ 0 L 1	eof	accept

(4 marks)

(ii) In the case of the Goto table, there are as many columns as non-terminal symbols, so no changes are expected. In the case of the Action table, when we have 2 lookahead symbols, we need to consider all possible combinations: ( ), ((,)),) (, (eof,) eof, eof, for a total of 7 columns. In the case of 3 lookahead symbols, we can easily calculate that we need a total of 15 combinations/columns. The key is to notice that some options are not possible, e.g., eof), etc.

(4 marks)

QUESTION 7:

As there are only 18278 different names, a possible approach is to compute a different value for each name and then perform a modulo 1024 operation. This would map about 18 different names to each symbol table entry, and assuming a good distribution for common names (e.g., single letter names) chances of collision would be minimized. Thus, a good hash function could be: [(ascii(LETTER1)-(ascii('a')+1) + (ascii(LETTER2)-(ascii('a')+1)*26 + (ascii(LETTER3)-(ascii('a')+1)*26*26] modulo 1024 (assuming a three letter name, LETTER1 LETTER2 LETTER3 – for shorter names, the corresponding components are removed). Any sensible answers that give a balanced distribution of names on the table will get full marks.

(5 marks)

QUESTION 8

When the execution reaches the printf statement for the first time, it has called the following functions: A(1) (from main), A(0) (from A), B(0) (from A), C(0) (from B). So, including main( ), there will be five activation records in the stack, each activation record corresponding to a function call.

(3 marks)

QUESTION 9:

Applying constant propagation first: the 2nd line becomes if (3>7)…, the 3rd line becomes …my_function(z*0), the 4th line becomes for(i=10; i<=5; … Applying dead-code elimination, the 2nd and 4th line disappear, applying constant folding the 3rd line becomes my_function(0). This results in:

```
b=4; c=1; d=3; n=5; scanf("%d",&z);
q=my_function(0);
printf("final %d \n",q);
```

Assuming that the variables b, c, d, n, z are not needed anywhere else in the code, the first line can be eliminated too. As required, students should show in each case the resulting code.

(6 marks)

QUESTION 10
The live ranges are:
r1 [1,4]   r2 [2,4]   r3 [3,6]   r1 [4,7]   r2 [5,8]   r3 [6,7]   r4 [7,8]   r5 [8,8]
(3 marks)

QUESTION 11:
The graph colouring algorithm may be based on ranking nodes according to the number of edges and then assigning a colour in some order taking care so that a neighbour does not share the same colour.
Number of edges per node:
A: 5, B: 2, C: 3, D: 3, E: 4, F: 2, G: 4, H: 4, I: 4, J:3
Then using the colours: Red-Green-Blue-Black:
A: Red
E: Red
G: Green
H: Green
I: Blue
C: Red
D: Blue
J: Black
B: Green
F: Green
The minimum number of colours needed is 4.
(4 marks)

QUESTION 12:
First build the precedence graph: (precedence graph not drawn here)
8 depends on 3 and 7; 7 depends on 5 and 2; 6 depends on 1 and 4; 5 depends on 4 and 1

Then assign weights; as latency is always 1 this is the longest path to exit:
1,4 have a weight of 4; 2,5 have a weight of 3; 3,7 have a weight of 2; 6,8 have a weight of 1

We schedule an instruction that is available, starting with those with highest weight:

Cycle1	1	nop	4
Cycle2	2	6	5
Cycle3	3	7	nop
Cycle 4	nop	nop	8

Or, in the case of 2 functional units:

Cycle1	1	4
Cycle2	2	5
Cycle3	3	7
Cycle4	8	6

Comparing the two schedules: (i) the length is the same; (ii) in the first case, we use cheaper functional units but utilisation is low.
(8 marks)