Ph.D. Qualification Exam 2019
Compiler Construction

1. Given the ***ac*** code segment, please write the corresponding Java assembly code using Jasmin Instructions, which are listed in Table 1 for your reference. In addition, the tokens defined in ***ac*** are listed in Table 2 to help you understand the ***ac*** code. (20pt)

   ***ac* code: f a f b i c a=1.8 c=9 b=9.9*a+c*3.2+1 a=b p a**

Table 1. List of Java assembly instructions.

Instruction	Functionality
iadd、*fadd*	add operation
isub、*fsub*	subtract operation
imul、*fmul*	multiple operation
idiv、*fdiv*	divide operation
ldc	load constant into stack; e.g., ldc 3
istore、*fstore*	store local variable
iload、*fload*	load local variable
getstatic、*putstatic*	Field manipulation instructions
invokevirtual、*invokestatic*、*invokespecial*	invoke methods
swap	exchange stack contents

Table 2. The tokens defined in ***ac*** language.

Terminal	Regular Expression
floatdcl	"f"
intdcl	"i"
print	"p"
assign	"="
plus	"+"
minus	"-"
multiply	"*"
div	"/"
inum	$[0-9]^+$
fnum	$[0-9]^+.[0-9]^+$
blank	$(" ")^+$
id	[a − e] | [g − h] | [j − o] | [q − z]

```
.class public main
.super java/lang/Object
.method public static main([Ljava/lang/String;)V
.limit stack 10 /* Define your storage size. */
.limit locals 3 /* Define your local space number. */

/* … (Answer) Java assembly code for the ac program … */

.end method
```

Note:

1) We assume that local variable ***0, 1,*** and ***2*** in the assembly code refer to the ***ac*** variable ***a, b,*** and ***c,*** respectively. Your answer (assembly code) should follow the assumption; **otherwise, it will be considered as wrong answer**.
2) A valid Java assembly program should include the code for the execution environment setup as above. You are excepted to answer WITHOUT listing the environment setup code.

2. Given the context-free grammar ***G***, an LL(1) grammar where ***X*** is start symbol, please write down the FIRST and FOLLOW sets for the non-terminals. In addition, please construct the parse table. Examples of your answers are listed as below. (20pt)

   Context-free grammar, ***G***.

```
1| X -> Yz
2|    | a
3| Y -> bZ
4|    | λ
5| Z -> λ
```

Example of your FIRST and FOLLOW sets.

Non-terminals	FIRST()	FOLLOW()
X		
Y		
Z		

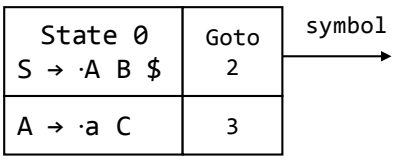
Example of your parse table.

Non-terminals /Terminals	a	b	z
X			
Y			
Z			

3. We know that a grammar, which is LL(1), may not be an LR(0) grammar sometimes.
   To determine if a grammar could be handled by the LR(0) parsing method, the common approach is to:
   1) build the transition diagram (i.e., characteristic finite-state machine, CFSM), and
   2) examine if there is any conflicts within each state of the diagram.
   We say that a grammar is not LR(0), if there is a **shift/reduce** or **reduce/reduce** conflict within a table entry. Otherwise, the grammar is LR(0).

   Here, we consider the LL(1) grammar, ***G***, from above again. You are ask to determine if ***G*** is LR(0) or not (i.e., a **Yes** or **No** answer) by building the transition diagram. If ***G*** is not LR(0), please indicate the exact conflict of the conflicting state. (20pt)

   Example of a transition diagram node.



   Note: reducible states are **double-boxed** nodes.

4. Given the code segment, ***K***, please draw their Control Flow Graphs (CFG). Note that code segment ***K*** is high-level (C-like) code. (20pt)

   Code segment, ***K***

```
while (...) {
      k = 2;
      if (...) {
            a = k + 2;
            x = 5;
      } else {
            a = k * 2;
            x = 8;
      }
      k = a;
      while (...) {
            b = 2;
            x = a + k;
            y = a * b;
            k++;
      }
      print(a + x);
}
print(a + k);
```

5. We have learnt several compiler optimization techniques, which transform the code into a more efficient form. Now, we consider the five optimization schemes: constant propagation, constant folding, copy propagation, loop unrolling, and unreachable code elimination. Please apply them to the four code segments, ***V, W, X,* and *Y***. (20pt)

   Note:

   1) One optimization technique should be applied to exactly a code segment; for example, if constant propagation were applied to code segment ***W***, it will not be applied to another code segment.
   2) Be advised that you should follow the example below to give your answer.

Code segment, ***V***

```
n = 10
c = 2
for (i = 0; i < n; i++) {
     s = s+i*c;
}
```

Code segment, ***W***

```
X = 1.9 * 2.5;
Y = 2.25 * 2.25;
Z = 8.7 * 8.7;
```

Code segment, ***X***

```
x = y;
if (x > 1) {
     s = x * f(x - 1);
}
```

Code segment, **Y**

```
if (a < c && false) {
        printf("I like compiler so much.\n");
        printf("I want to attend this exam again next year\n");
        printf("☺☺☺\n");
} else {
        printf("I would like to pass this exam this time.\n")
}
```

Example of your answer for code transformation.

Code segment, ***A*** applies common subexpression elimination.

```
a = b + c;   →   a = b + c;
c = b + c;       c = a;
d = b + c;       d = b + c;
```