Name: Student ID:	Compiler Construction, Spring 2022 Final (120pt)

1. (16pt) Please answer the following questions.

   a) What is the difference between a **compiler** and an **interpreter**? (4pt)

   a)

   - Interpreter:
     - Interpreter **directly interprets (executes)** the source program that reads inputs and writes outputs
     - Interpreters execute programs **without explicitly performing much translation**
   - compiler:
     - Answer1:
       - The compilation phase **generates target program from source program** (i.e., compiler does the translation)
       - The execution phase **executes the target program** (i.e., user run the translated program)
     - Answer2:
       - A compiler is a program that
         - accepts, as input, a program text in a certain programming language (source language), and
         - produces, as output, a program text in an assembly language (target language),
         - which will later be assembled by the assembler into machine code

   b) What is the benefit of **Bottom-Up Parsing** compared with **Top-Down Parsing**? (4pt)

   Bottom-up parsers are power, efficiency, and ease of construction. Bottom-up parsers can handle the largest class of grammars that allow parsing to proceed deterministically

   c) Please describe the main goal of the **LLVM Link-Time Optimization**. (4pt)

   The LLVM Link Time Optimizer provides complete transparency, while doing intermodular optimization, in the compiler tool chain. Its main goal is to let the developer take advantage of intermodular optimizations without making any significant changes to the developer's makefiles or build system.

d) The load instruction in the JVM is not zero-address. Please describe the **pure zero-address form problem** and how JVM deals with the load instruction. (4pt)

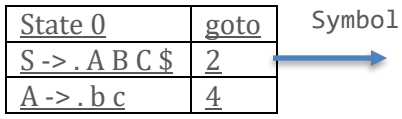
The registers that could be accessed by a load instruction may not be known until runtime. Runtime checks could be deployed to check the validity of a load instruction but degrades performance.
JVM specifies the register number as an immediate operand of the instruction.

2. (35pt) Given the grammar ***G***, please answer the following questions.
   a) Please construct the transition diagram for the grammar ***G***. (10pt)
   b) According to a), ***G*** is not an LR (0) grammar. Please indicate what kinds of the conflict occur and in which state. (5pt)
   c) Please construct SLR (1) parse table for the grammar ***G***. (10pt)
   d) Based on c), please show the parsing procedure for the input string: c a d a a $. (10pt)

Context-free grammar ***G***:

```
1| S -> A C B $
2| A -> C
3|    | a c
4| B -> B a
5|    | a
6| C -> B d
7| C -> c
```

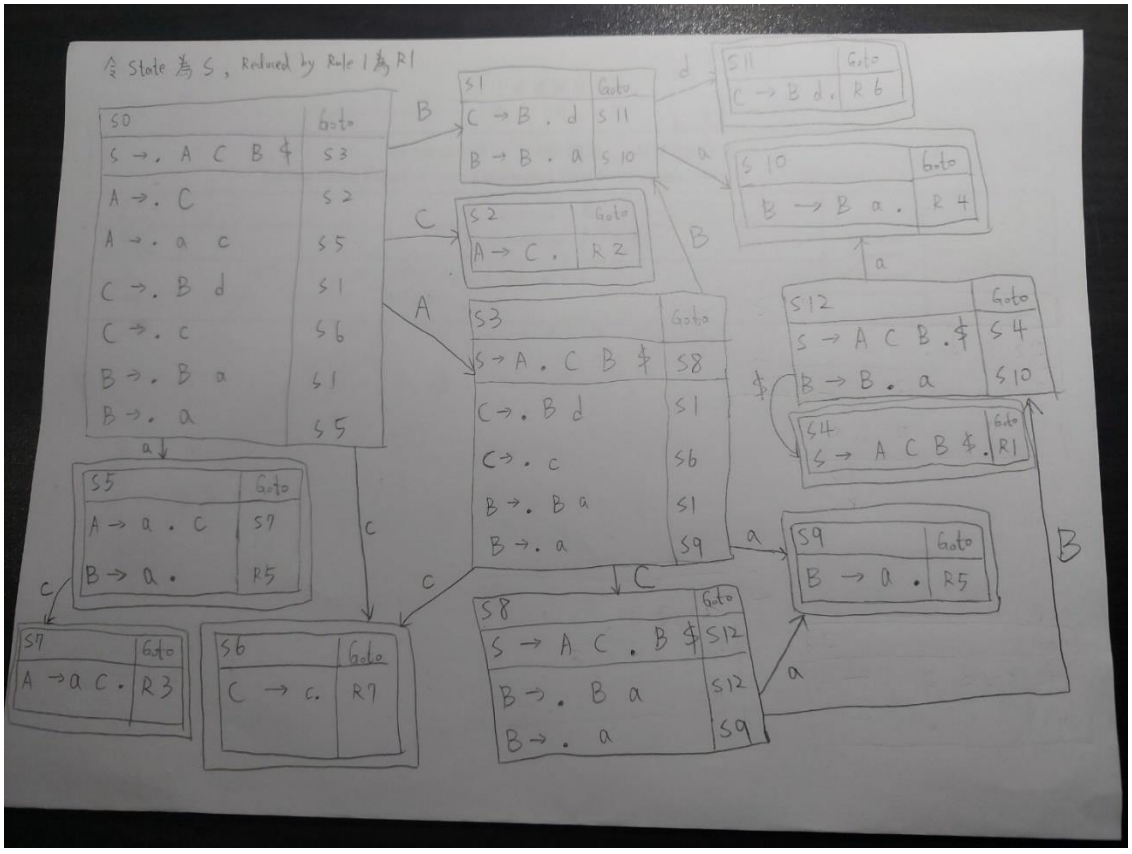
Format example of a transition diagram node



Format example of the parsing procedure

Step	Stack	Remaining Input	Action
1	0	a b $	Shift a
2	0 a1	b $	Reduced by Rule3

a) 少 symbol 跟 goto 哪個 state，都會稍微扣分，因題目有說明。



b) shift/reduce conflict in state 5.

會根據同學畫的 a 小題來評分。

c) 本題是問 SLR (1) parse table，注意該 reduce 的 symbol，不會全都是整列。

	$	c	d	a	S	A	C	B
0		S6		S5	Accept	S3	S2	S1
1			S11	S10				
2		R2		R2				
3		S6		S9			S8	S1
4	R1							
5	R5	S7	R5	R5				
6		R7		R7				
7		R3		R3				
8				S9				S12
9	R5		R5	R5				
10	R4		R4	R4				
11		R6		R6				
12	S4			S10				

d) stack 中沒寫 state 會扣分，題目有給範例說明。

input string: c a d a a $

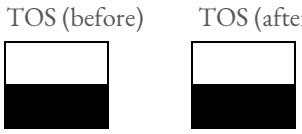
Step	STACK	Remaining Input	Action	
1	0	c a d a a $	Shift c	
2	0 c6	a d a a $	Reduced by Rule 7| C -> c	
3	0	C a d a a $	Shift C	
4	0 C2	a d a a $	Reduced by Rule 2| A -> C	
5	0	A a d a a $	Shift A	
6	0 A3	a d a a $	Shift a	
7	0 A3 a9	d a a $	Reduced by Rule 5| B -> a	
8	0 A3	B d a a $	Shift B	
9	0 A3 B1	d a a $	Shift d	
10	0 A3 B1 d11	a a $	Reduced by Rule 6| C -> B d	
11	0 A3	C a a $	Shift C	
12	0 A3 C8	a a $	Shift a	
13	0 A3 C8 a9	a $	Reduced by Rule 5| B -> a	
14	0 A3 C8	B a $	Shift B	
15	0 A3 C8 B12	a $	Shift a	
16	0 A3 C8 B12 a10	$	Reduced by Rule 4| B -> B a	
17	0 A3 C8	B $	Shift B	
18	0 A3 C8 B12	$	Shift $	
19	0 A3 C8 B12 $4		Reduced by Rule 1| S -> A C B $	
20	0	S	Accept	Shift S

3. (25pt) Please answer the following question.

   a) 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**

   b) Based on a), please select one instruction of your Java assembly code. Next, draw its stack before and after the instruction. (5pt)



任意畫其中一個指令的 stack 即可。

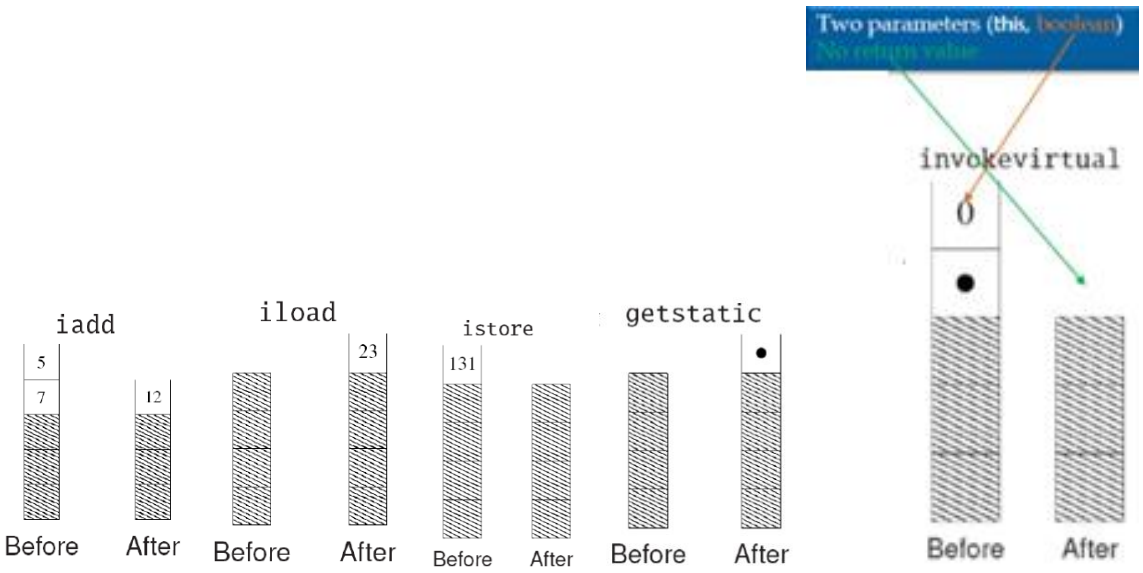


Table 1. List of Java assembly instructions. language.

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***

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 expected to answer WITHOUT listing the above environment setup code.
3) Please consider the precedence that "*" > "+".

***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**

```
ldc 0.0
fstore 0
ldc 0.0
fstore 1
ldc 0
istore 2
ldc 1.8
fstore 0
ldc 9
istore 2
ldc 9.9
fload 0
fmul
iload 2
i2f
ldc 3.2
fmul
fadd
ldc 1
i2f
fadd
fstore 1
fload 1
fstore 0
fload 0
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/println(F)V
```

i2f 沒寫不扣分，因題目的參考一開始沒給。

4. (24 pt) Here is a list of optimizations.

a) Constant Propagation	b) Constant Folding	c) Algebraic Simplification
d) Copy Propagation	e) Common Subexpression Elimination	f) Dead Code Elimination
g) Loop Unrolling	h) Strength Reduction	i) Function Inling
j) Function Cloning	k) Expression Propagation	

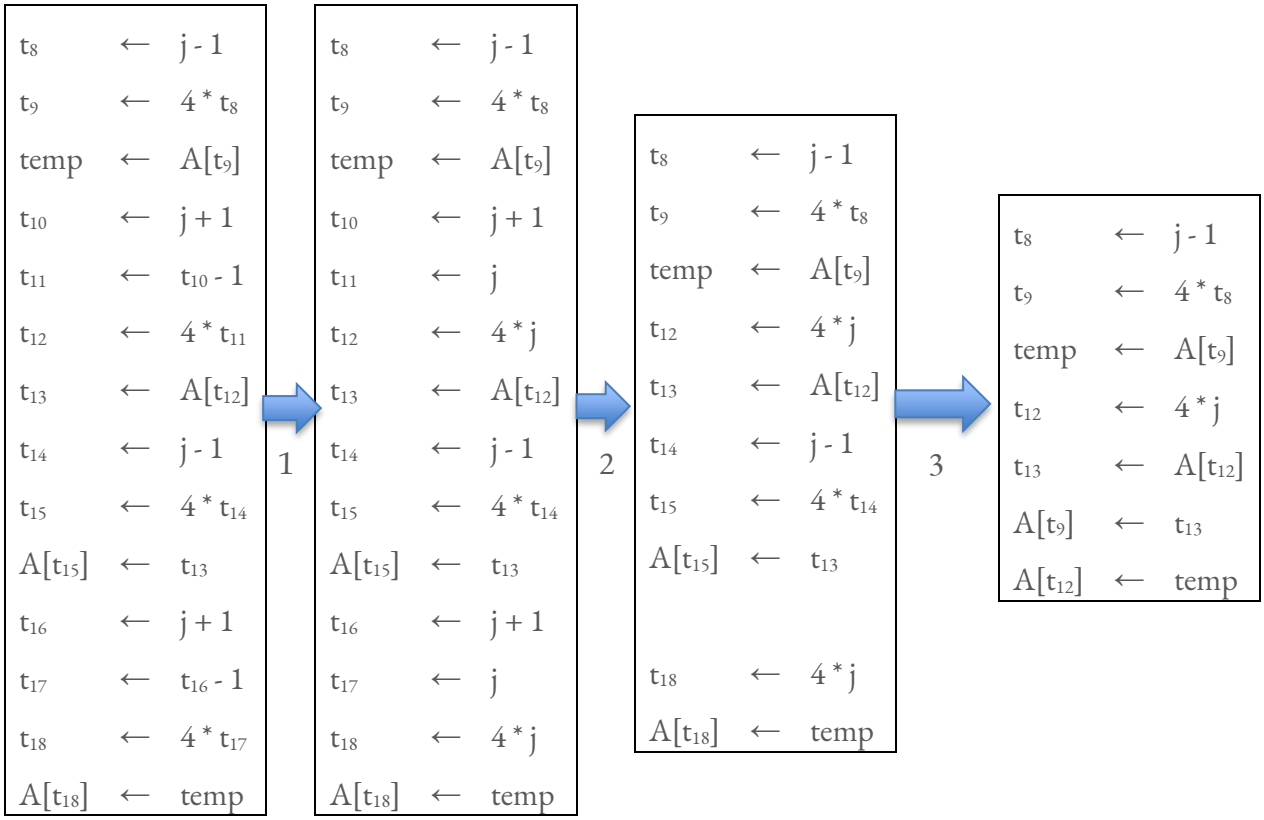
Hint for **k) Expression Propagation**:

Before	After
$a_1 \leftarrow k + 1$	$a_1 \leftarrow k + 1$
$a_2 \leftarrow a_1 - 1$	$a_2 \leftarrow k$

Consider the following section of a flow-graph for a bubble-sort algorithm in three-address code:

t_8	$\leftarrow$	$j - 1$
t_9	$\leftarrow$	$4 * t_8$
temp	$\leftarrow$	$A[t_9]$
t_{10}	$\leftarrow$	$j + 1$
t_{11}	$\leftarrow$	$t_{10} - 1$
t_{12}	$\leftarrow$	$4 * t_{11}$
t_{13}	$\leftarrow$	$A[t_{12}]$
t_{14}	$\leftarrow$	$j - 1$
t_{15}	$\leftarrow$	$4 * t_{14}$
$A[t_{15}]$	$\leftarrow$	t_{13}
t_{16}	$\leftarrow$	$j + 1$
t_{17}	$\leftarrow$	$t_{16} - 1$
t_{18}	$\leftarrow$	$4 * t_{17}$
$A[t_{18}]$	$\leftarrow$	temp

The following is the evolution of the basic block through several optimization stages:



For each optimization stage, identify all optimization(s) applied (there can be more than one).

每個 stage，多 1 少 1 扣 2 分。

Stage 1: (8pt) **b or c, d, k**

- **k) Expression Propagation** : $t_{11} \leftarrow t_{10} - 1$ becomes $t_{11} \leftarrow j + 1 - 1$, similarly for t_{17}
- **b) Constant Folding** or **c) Algebraic Simplification** : $t_{11} \leftarrow j + 1 - 1$ becomes $t_{11} \leftarrow j$, similarly for t_{17}
- **d) Copy Propagation** : $t_{12} \leftarrow 4 * t_{11}$ becomes $t_{12} \leftarrow 4 * j$, similarly for t_{18}

Stage 2: (8pt) **f**

- **f) Dead Code Elimination** : remove $t_{10} \leftarrow j + 1$, $t_{11} \leftarrow j$, $t_{16} \leftarrow j - 1$, $t_{17} \leftarrow j$

Stage 3: (8pt) **d, e, f**

- **e) Common Subexpression Elimination** : $t_{14} \leftarrow j - 1$ becomes $t_{14} \leftarrow t_8$, and $t_{18} \leftarrow 4 * j$ becomes $t_{18} \leftarrow t_{12}$

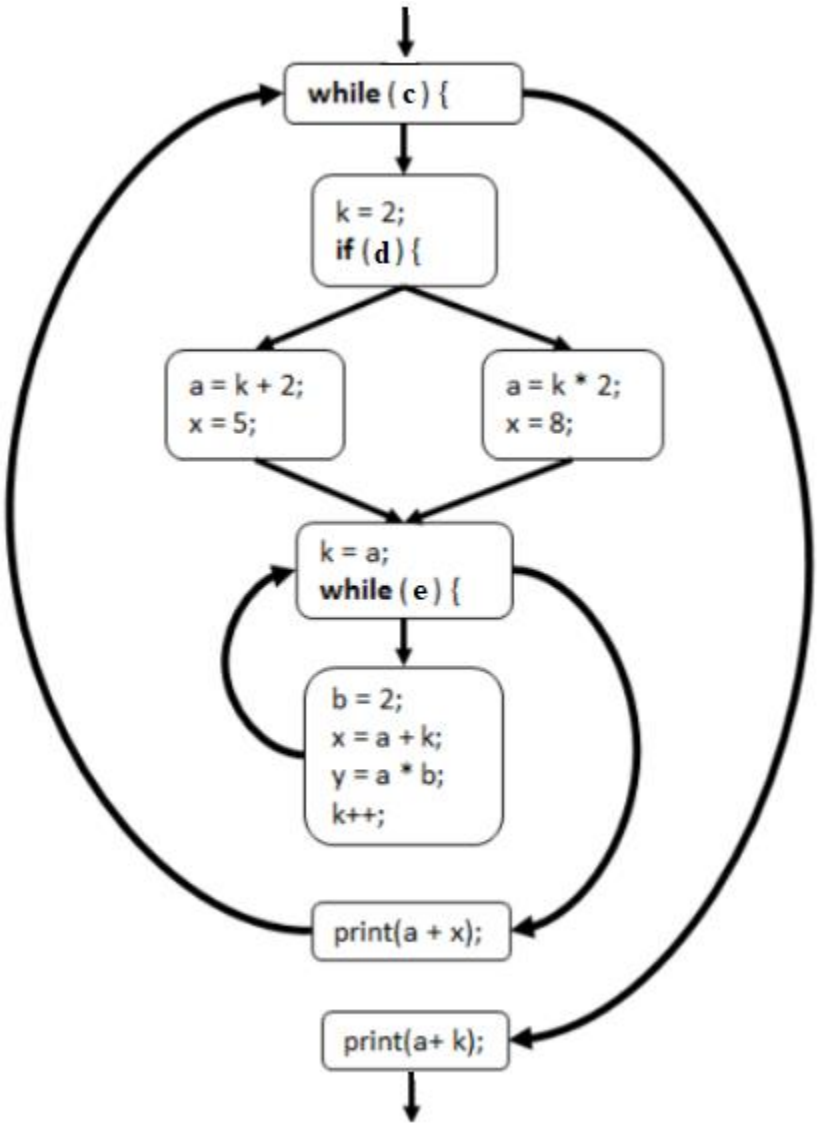
- **d) Copy Propagation** : $t_{15} \leftarrow 4 * t_{14}$ becomes $t_{15} \leftarrow 4 * j$
- **e) Common Subexpression Elimination** : $t_{15} \leftarrow 4 * j$ becomes $t_{15} \leftarrow t_{19}$
- **d) Copy Propagation :** $A[t_{15}] \leftarrow t_{13}$ becomes $A[t_9] \leftarrow t_{13}$, and $A[t_{18}] \leftarrow$ temp becomes $A[t_{12}] \leftarrow$ tempa
- **f) Dead Code Elimination** : remove $t_{14} \leftarrow t_8$, $t_{15} \leftarrow t_9$, $t_{18} \leftarrow t_{12}$,

5. (20 pt) Given the code segment I, please answer the following question.

   a) Please draw the **Control Flow Graph (CFG)** for code segment I. (20 pt)

Code segment I:

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


while (c) {
k = 2;
if (d) {
a = k + 2;
x = 5;
a = k * 2;
x = 8;
k = a;
while (e) {
b = 2;
x = a + k;
y = a * b;
k++;
print(a + x);
print(a+ k);