Chapter 1

Exercises 1.1

1. Show *assembly language* for a machine of your choice, corresponding to each of the following Java statements:

(a)
```
a = b + c;
```

```
ld      r1,b
add     r1,c
sto     r1,a
```

(b)
```
a = (b+c) * (c-d);
```

```
ld      r1,b
add     r1,c
ld      r2,c
sub     r2,d
mr      r1,r2
sto     r1,a
```

ld = LOD

brh = BH

ar = Add

mr = mov

(c)
```
for (i=1; i<=10; i++) a = a+i;
ld      r1,1
ld      r2,a
loop:
cmp     r1,='10'
brh     done
ar      r2,r1
incr    r1
jmp     loop
done:
```

2. Show the difference between compiler output and interpreter output for each of the following source inputs:

(a)
```
a = 12;
b = 6;
c = a+b;
println (c,a,b);

Compiler output:
     mov  a,='12'
          mov  b,='6'
          lod  r2,a
          lod  r3,b
          lod  r1,a
          ar   r1,r3
          sto  r1,c
          stm  r1,r3,parms
          call println

Interpreter output:
          18126
```

```
mov a, ='12'
mob b,='6'
lod r1,a
lod r2,b
lod r3,a
add r3,r2
sto r3,c
push r3
push r2
push r1
call print
```

(b)
```
a = 12;
     b = 6;
     if (a<b) println (a);
     else println (b);
```

Compiler output:

```
        lod     r1,='12'
        lod     r2,='6'
        cmpr    r1,r2
        bge     less
        st      r1,parms
        call    println
        jmp     out
        less:
        st      r2,parms
        call    println
        out:
```

Interpreter output:

```
        6
```

(c)

```
a = 12;
b = 6;
while (b<a)
     {  a = a-1;
        println (a+b);
     }
```

```
Compiler output:
     lod  r1,='12'
     lod  r2,='6
     loop:
     cmpr r1,r2
     bge  done
     decr r1
     stm  r1,r2,parms
     call println
     jmp  loop
     done:

Interpreter output:
     116
     106
```

```
96
86
76
66
```

3. Which of the following Java source errors would be detected at compile time, and which would be detected at run time?

(a)
```
a = b+c = 3;
```
Compile time error

(b)
```
if (x<3) a = 2
      else a = x;
```
Compile time error

(c)
```
if (a>0) x = 20;
     else if (a<0) x = 10;
          else x = x/a;
```
Run time error

(d)
```
MyClass x [] = new MyClass[100];
x[100] = new MyClass;
```
Run time error

4. Using the big C notation, show the symbol for each of the following:

(a) A compiler which translates COBOL source programs to PC machine language and runs on a PC.

(b) A compiler, written in Java, which translates FORTRAN source programs to Mac machine language.

(c) A compiler, written in Java, which translates Sun machine language programs to Java.

1.2 The Phases of a Compiler

Exercises 1.2

1. Show the *lexical tokens* corresponding to each of the following Java source inputs:

(a)
```
for (i=1; i<5.1e3; i++) func1(x);
```

keyword	for
special char	(
identifier	i
operator	=
numeric const	1
special char	;
identifier	i
operator	<
numeric const	5.1e3
special char	;
identifier	i
operator	++
special char	)
identifier	func1
special char	(
identifier	x
special char	)
special char	;

(b)
```
if (sum!=133) /* sum = 133 */
```

keyword	if
special char	(
identifier	sum
operator	!=
numeric const	133
special char	)
comment	/* sum = 133 */

(c)
```
) while ( 1.3e-2 if &&
```

special char	)
keyword	while

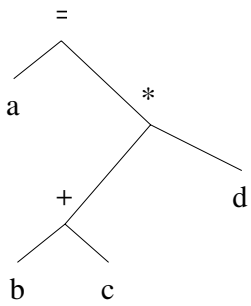
special char	(
numeric const	1.3e-2
keyword	if
operator	&&

(d)
```
if 1.2.3 < 6
```

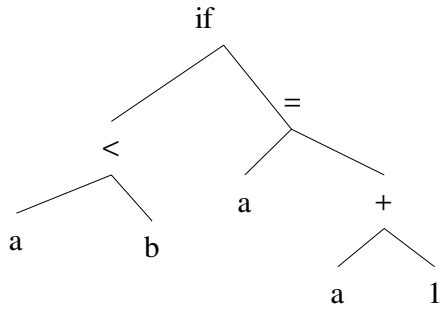
keyword	if
numeric const	1.2
numeric const	.3
operator	<
numeric const	6

2. Show the sequence of atoms put out by the parser, and show the *syntax tree* corresponding to each of the following Java source inputs:

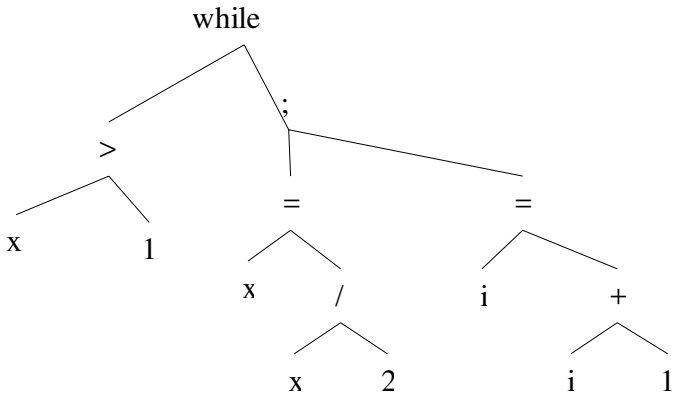
(a)
```
a = (b+c) * d;
```
(ADD, b, c, T1)
(MUL, T1, d, T2)
(MOV, T2, , a)



(b)
```
if (a<b) a = a + 1;
```
(TST, a, b, , 2, L1)
(JMP, L2)
(LBL, L1)
(ADD, a, 1, T1)
(MOV, T1, , a)
(LBL, L2)

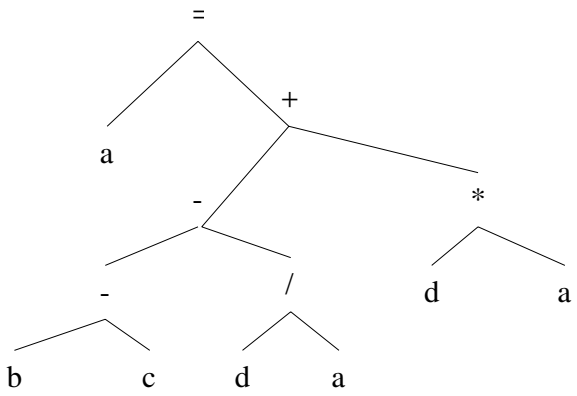


(c)
```
while (x>1)
    { x = x/2;
      i = i+1;
    }
```
(LBL, L1)
(TST, x, 1, 3, , L3)
(JMP, L2)
(LBL, L3)
(DIV, x, 2, T1)
(MOV, T1, , x)
(ADD, i, 1, T2)
(MOV, T2, , i)
(JMP, L1)
(LBL, L2)



(d) `a = b - c - d/a + d * a;`

(SUB, b, c, T1)
(DIV, d, a, T2)
(SUB, T1, T2, T3)
(MUL, d, a, T4)
(ADD, T3, T4, T5)
(MOV, T5, , a)



3. Show an example of a *Java statement* which indicates that the order in which the two operands of an ADD are evaluated can cause different results:

```
operand1 + operand2
```

(a=2) + (a=3)

method1() + method2()

4. Show how each of the following *Java source inputs* can be optimized using global optimization techniques:

(a)
```
for (i=1; i<=10; i++)
    { x = i + x;
      a[i] = a[i-1];
      y = b * 4;
    }
```

```
for (i=1; i<=10; i++)
    { x = i + x;
      a[i] = a[i-1];
    }
  y = b * 4;
```

(b)
```
for (i=1; i<=10; i++)
    { x = i;
      y = x/2;
      a[i] = x;
    }
```

```
for (i=1; i<=10; i++)
     a[i] = i;
x = 10;
y = x/2;
```

(c)
```
if (x>0) {x = 2; y = 3;}
```

```
        else {y = 4; x = 2;}

if (x>0)
     y = 3;
else
     y = 4;
x = 2;
```

(d)
```
     if (x>0) x = 2;
     else if (x<=0) x = 3;
     else x = 4;

     if (x>0) x = 2;
     else  x = 3;
```

5. Show, in *assembly language* for a machine of your choice, the output of the code generator for the following atom string:

```
(ADD,A,B,Temp1)
(SUB,C,D,Temp2)
(TEST,Temp1,<,Temp2,L1)
(JUMP,L2)
(LBL,L1)
(MOVE,A,B)
(JUMP,L3)
(LBL,L2)
(MOVE,B,A)
(LBL,L3)
```

```
lod  r1,A
add  r1,B
sto  r1,Temp1
lod  r1,C
sub  r1,D
sto  r1,Temp2
jmp  L2
L1:
```

```
lod   r1,A
sto   r1,B
jmp   L3
L2:
lod   r1,B
sto   r1,A
L3:
```

6. Show a *Java source statement* which might have produced the atom string in Problem 5, above.

```
if (A+B < C-D) A = B; else B = A;
```

7. Show how each of the following *object code segments* could be optimized using local optimization techniques:

(a)

```
LD   R1,A
MULT R1,B
ST   R1,Temp1
LD   R1,Temp1
ADD  R1,C
ST   R1,Temp2
```

```
LD   R1,A
MULT R1,B
ADD  R1,C
ST   R1,Temp2
```

(b)

```
LD   R1,A
ADD  R1,B
ST   R1,Temp1
MOV  C,Temp1
```

```
LD   R1,A
ADD  R1,B
ST   R1,C
```

(c)

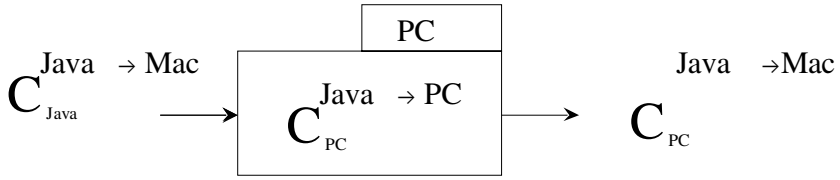
```
        CMP   A,B
        BH    L1
        B     L2
L1:     MOV   A,B
        B     L3
L2:     MOV   B,A
L3:
```

```
        CMP   A,B
        BLE   L1
        MOV   A,B
        B     L3
L1:     MOV   B,A
L3:
```

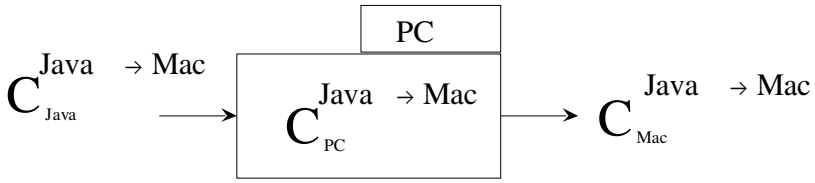
Exercises 1.3

1. Fill in the missing information in the compilations indicated below:

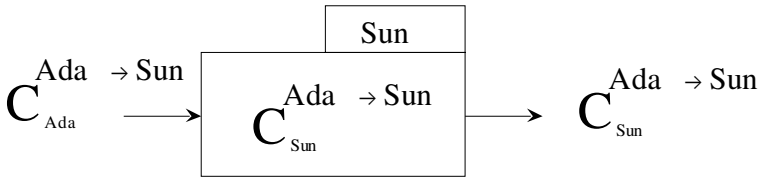
(a)



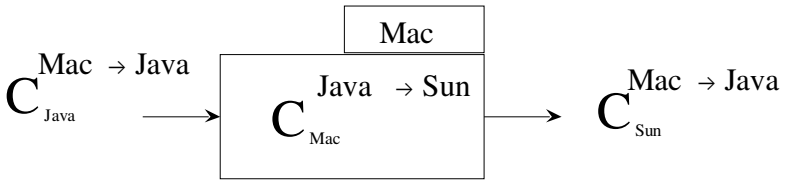
(b)



(c)



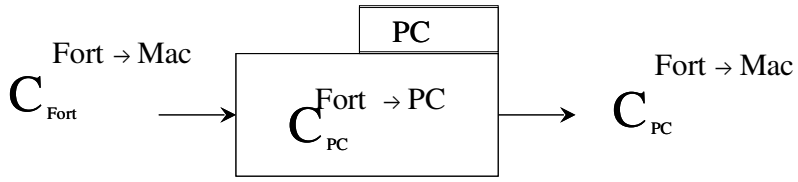
(d)

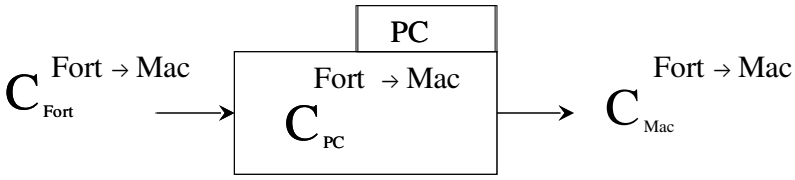


2. How could the compiler generated in part (d) of Question 1 be used?

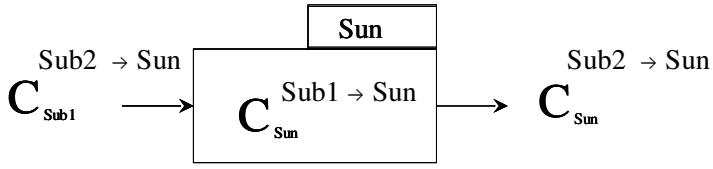
It could be used to decompile Mac programs (i.e. executables) to Java, using a Sun computer.

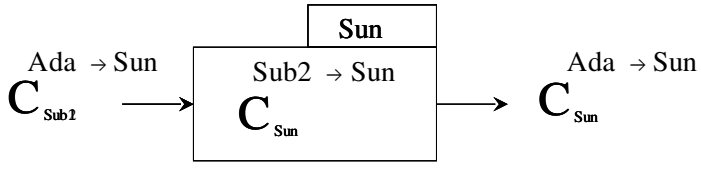
3. If the only computer you have is a PC (for which you already have a FORTRAN compiler), show how you can produce a FORTRAN compiler for the Mac computer, without writing any assembly or machine language.





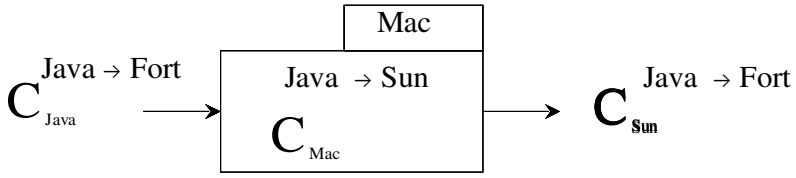
4. Show how Ada can be bootstrapped in two steps on a Sun, using first a small subset of Ada, `Sub1`, and then a larger subset, `Sub2`. First use `Sub1` to implement `Sub2` (by bootstrapping), then use `Sub2` to implement Ada (again by bootstrapping). `Sub1` is a subset of `Sub2`.





5. You have 3 computers: a PC, a Mac, and a Sun. Show how to generate automatically a Java to FORT translator which will run on a Sun if you also have the four compilers shown below:

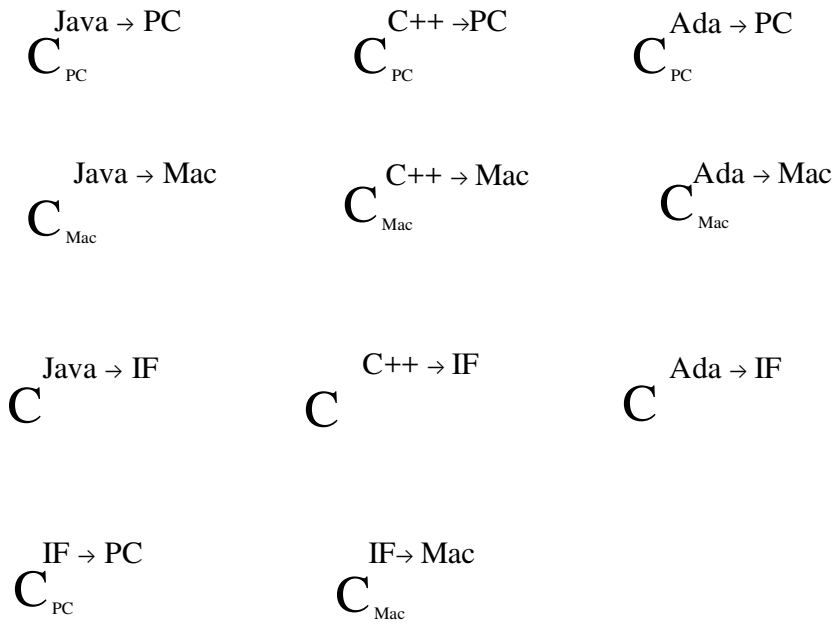
$C^{\text{Java} \to \text{Fort}}_{\text{Mac}}$ $C^{\text{Fort} \to \text{Java}}_{\text{Sun}}$ $C^{\text{Java} \to \text{Sun}}_{\text{Mac}}$ $C^{\text{Java} \to \text{Fort}}_{\text{Java}}$



6. In Figure 1.8 suppose we also have $C^{\text{Java} \to \text{Sun}}_{\text{Java}}$. When we write $C^{\text{Java} \to \text{Mac}}_{\text{Java}}$, which of the phases of $C^{\text{Java} \to \text{Sun}}_{\text{Java}}$ can be reused as is?

Lexical and Syntax (also Global Optimization)

7. Using the big C notation, show the 11 translators which are represented in figure 1.9. Use "Int" to represent the intermediate form.



Exercises 1.4

1. Which of the following are valid program segments in Decaf? Like Java, Decaf programs are free-format (Refer to Appendix A).

(a)
```
for (x = 1; x<10; )
     y = 13;
```

Valid

(b)
```
if (a<b) { x =
     2;  y = 3 ;}
```

Valid