上 海 交 通 大 学 试 卷（A 卷）

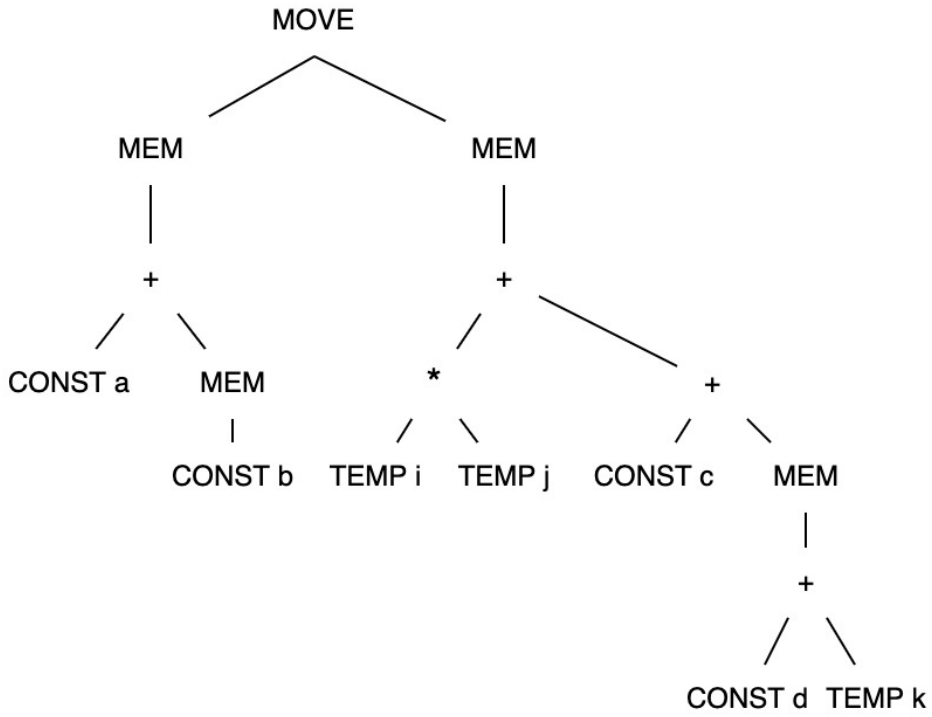
（ 2022 至 2023 学年 第 1 学期 ）

班级号_____ 学号_____ 姓名 _____

课程名称_____编译原理与技术_____成绩 _____

Problem 1: Instruction Selection (24 points)

1.

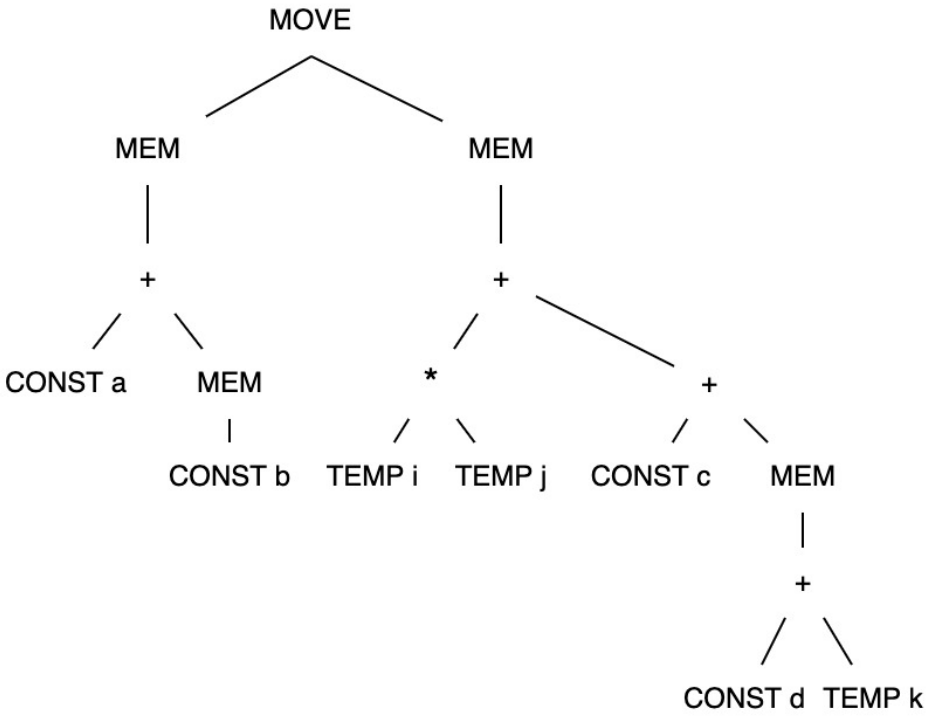


我承诺，我将严格遵守考试纪律。

题号									
得分									
批阅人(流水阅卷教师签名处)									

承诺人：

2.



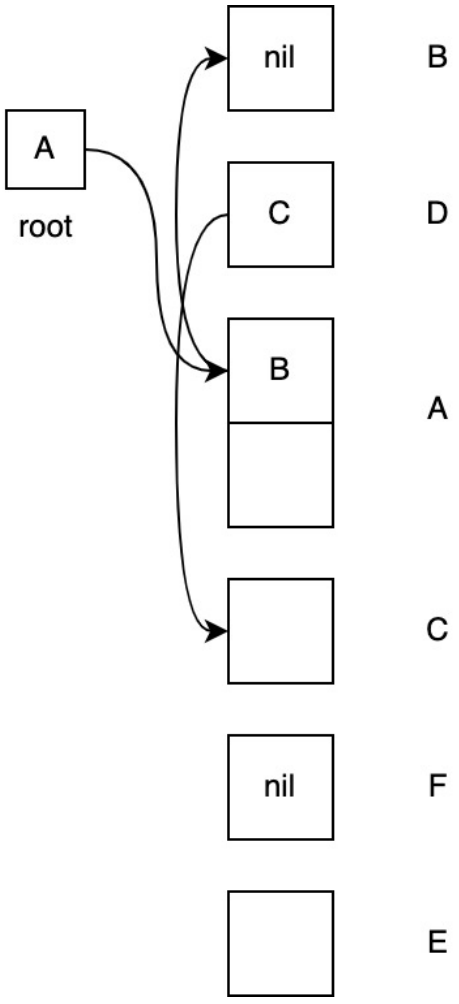
Problem 2: Garbage Collection (24 points)

1.

2.

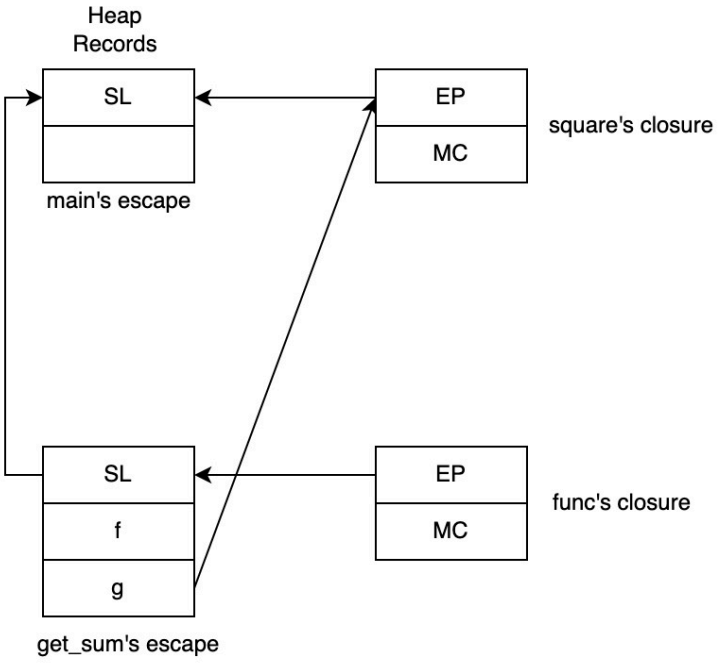
3.

4.



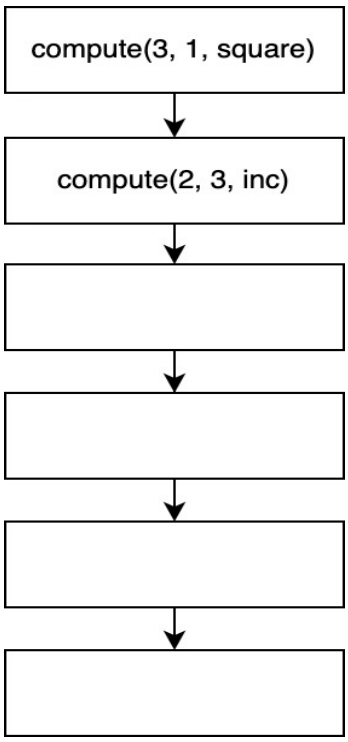
Problem 3: Functional Programming (20 points)

1.



2.

3.



Problem 4: Register Allocation (32 points)

1.

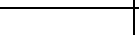
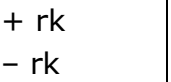
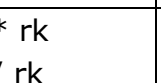
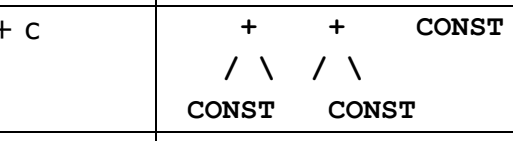
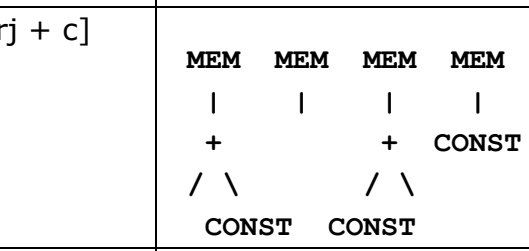
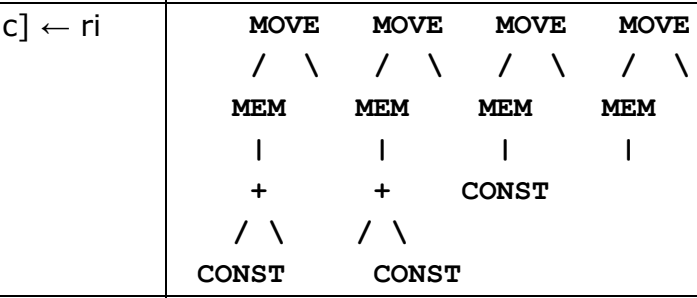
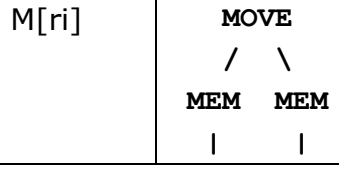
2.

instr	def	use	in	out
t1 <- r1				
t2 <- r2				
i <- r3				
j <- 1				
jmp L3				
r3,r4 <- i / j				
cmp r4, 0				
jne L2				
r3 <- j				
call print				
j <- j + 1				
cmp j, i				
jl L1				
r2 <- t2				
r1 <- t1				
ret				

3.

4 .

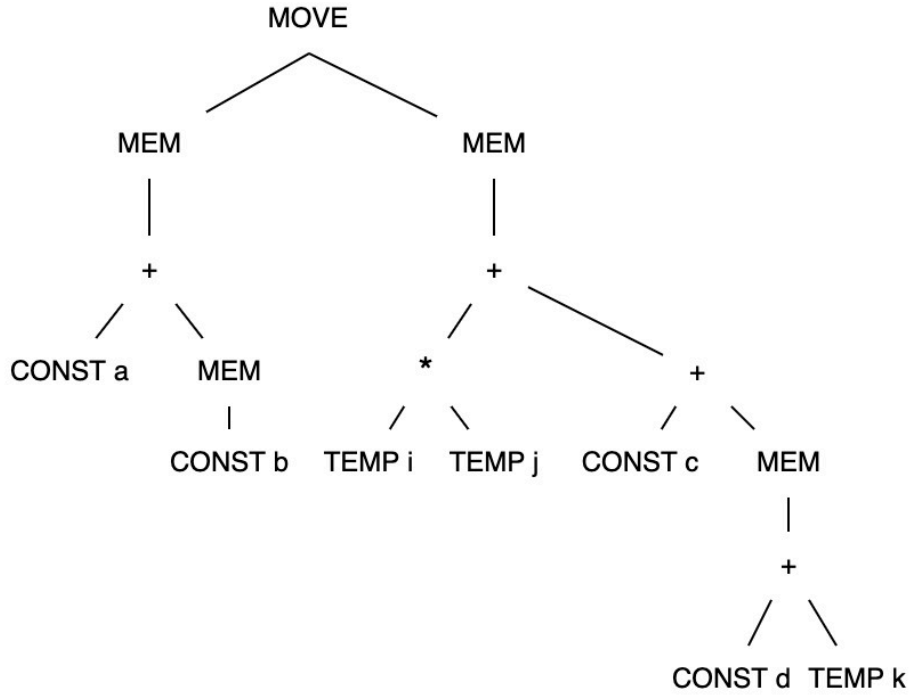
Problem 1: Instruction Selection (24 points)

Name	Effect	Trees	Cost
-	ri		1
ADD SUB	ri ← rj + rk ri ← rj – rk		2
MUL DIV	ri ← rj * rk ri ← rj / rk		2
ADDI	ri ← rj + c		3
LOAD	ri ← M[rj + c]		4
STORE	M[rj + c] ← ri		4
MOVEM	M[rj] ← M[ri]		4

Note:

- The notation **M[x]** denotes the memory word at address x.
- The **third column** in the table above is the set of **tiles** corresponding to the machine instruction in the first column of the table
- The **fourth column** in the table above is the **cost** corresponding to the machine instruction in the first column of the table
- On this architecture, a register r_0 always contains zero

Consider the following IR tree and answer the questions below.



1. In the course, we have learnt **maximal munch algorithm** to tile the IR tree. Please use this algorithm to tile the IR tree above. **Draw your tiles**, **write the corresponding instruction sequences** after tiling, and **calculate the cost** of your instruction sequences. (12')

2. We also have learnt a **dynamic programming algorithm** to find the **optimum** tiling to the IR tree. Please use the dynamic-programming algorithm to tile the IR tree above. **Draw your tiles, write the corresponding instruction sequences** after tiling, and **calculate the cost** of your instruction sequences. (12')

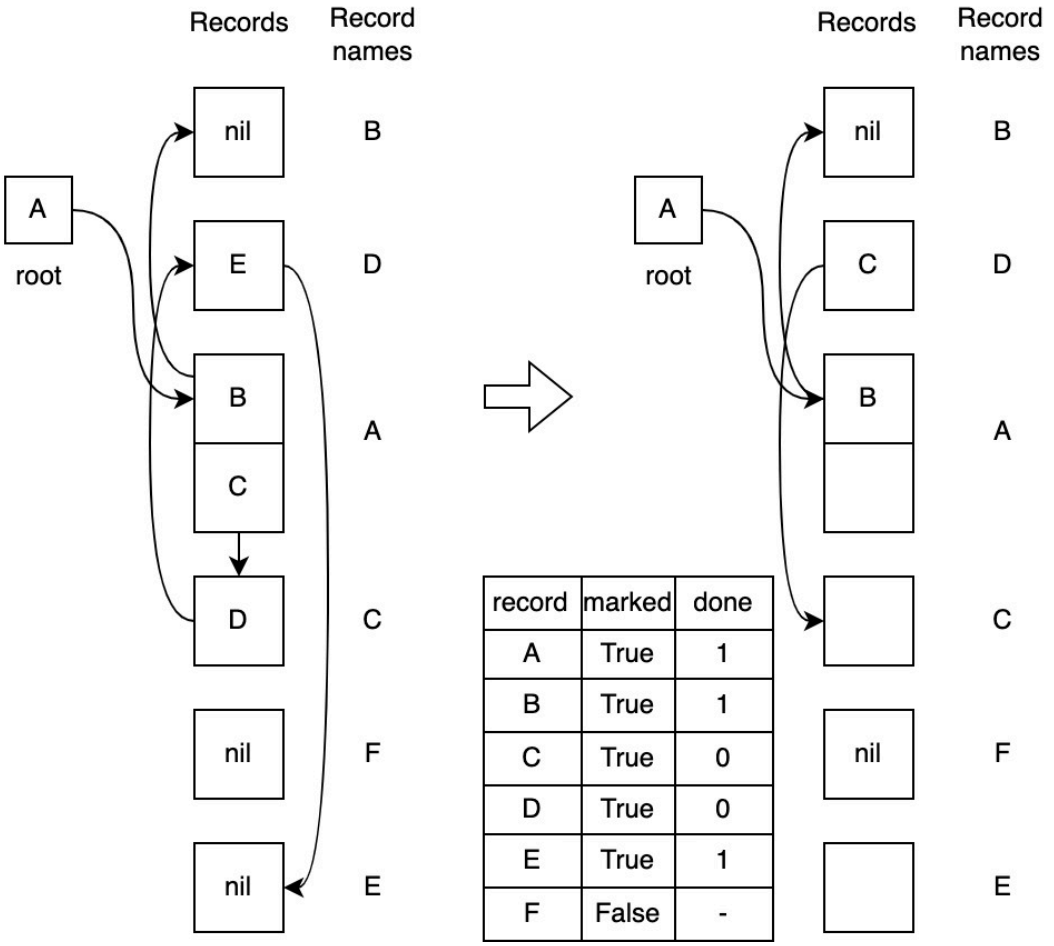
Problem 2: Garbage Collection (24 points)

1. **Reference counting** may not be able to identify all garbages on heap. Why? (4')

2. In **generational collection**, if we want to do garbage collection in generation G1, what should we do to handle pointers from roots in G0 (younger than G1) and G2 (older than G1)? (5')

3. When using mark-and-sweep collection or copying collection, we need to **find the roots** (program variables that are **pointers**) in the program before starting DFS or BFS. How can garbage collectors know **which program variables are pointers**? (5')

4. In the mark-and-sweep collection, we can use **pointer reversal** to avoid large auxiliary stack memory due to **DFS (depth-first search)**. The left diagram shows the **initial state of the heap**, and the right diagram and table show **an intermediate state in a DFS**. Please complete **the right diagram** according to the algorithm of pointer reversal. You need to fill in fields and draw arrows to make it correct. You can refer to **Algorithm 13-4 in the Chinese textbook** or **Algorithm 13.6 in the English textbook**. (10')

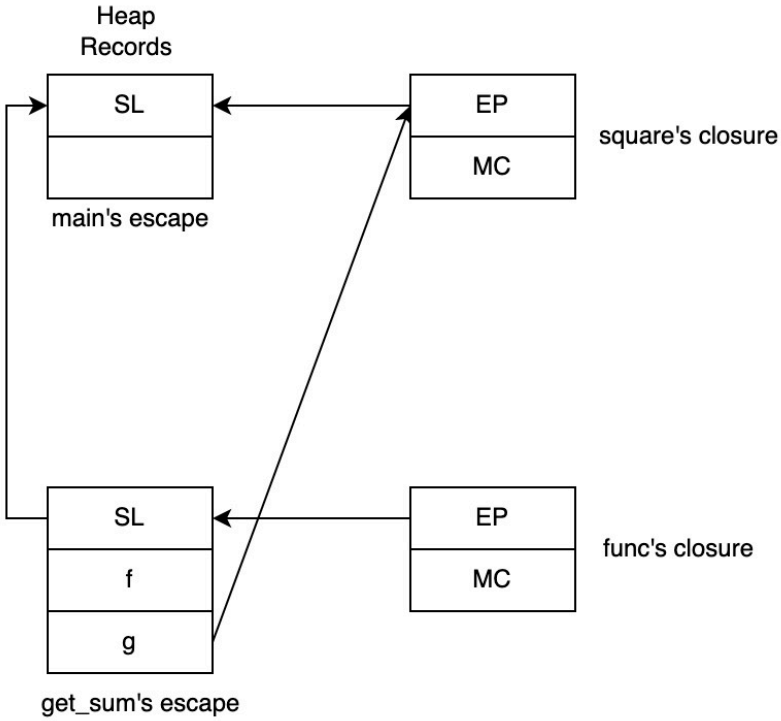


record	marked	done
A	True	1
B	True	1
C	True	0
D	True	0
E	True	1
F	False	-

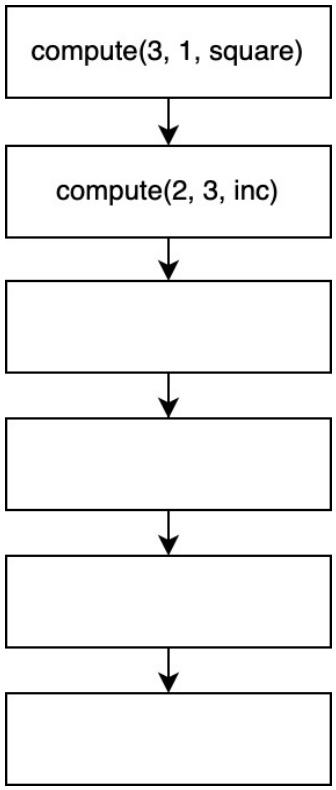
Problem 3: Functional Programming (20 points)

```
1  let
2     type intfun = int -> int
3
4     function multiply(n: int) : intfun =
5        let function h(m: int) : int = m * n
6         in h
7        end
8
9     function square(a: int) : int = a * a
10
11    function get_sum(f: intfun, g: intfun) : intfun =
12       let function func(x: int, y: int) : int = f(x) + g(y)
13        in func
14       end
15
16    var x := multiply(2)
17
18    function compute(i: int, res: int, c: intfun) =
19       if i=1 then c(res)
20       else let function inc(b: int) = c(b + 1)
21             in compute(i-1, res * i, inc)
22            end
23
24 in
25    var eval = get_sum(x, square)
26    var sum = eval(-3, 3)
27    print(sum)
28    compute(3, 1, square)
29 end
```

1. When function **get_sum** is **returned** in **line 25**, part of the allocated activation records is shown in the following figure. Please **draw the missing activation record(s) and closure(s)**, and **add arrows** to indicate the relation of them. (8')



2. What’s the actual value of **sum** in **line 26**? (4’)
3. Please complete the following control flow graph for **all the function calls** invoked within **line 28**. Every blank should be filled with the current function call with **the actual value of its arguments**. (8’)



Problem 4: Register Allocation (32 points)

Suppose a compiler has compiled the following function **find_factor** into instructions on the right:

```
void find_factor(int i) {
    int j = 1;
    while (j < i) {
        if (i % j == 0)
            print(j);
        j++;
    }
}
```

```
find_factor:
    t1 <- r1
    t2 <- r2
    i <- r3
    j <- 1
    jmp L3
L1:
    r3,r4 <- i / j
    cmp r4, 0
    jne L2
    r3 <- j
    call print
L2:
    j <- j + 1
L3:
    cmp j, i
    jl L1
    r2 <- t2
    r1 <- t1
    ret
```

Several things are worth noting in the instructions:

- There are **four** hardware registers in all.
- The compiler passes parameters using registers. (The first parameter goes to **r3**)
- **r1,r2** is **callee-saved** while **r3,r4** are **caller-saved**.
- The architecture handles integer divisions by fetching the two operands from **arbitrary** registers. After calculation, it stores the **quotient** and **remainder** to **r3** and **r4** respectively.
- **t1, t2, i, j** are temporary registers.

Please answer the following questions according to the instructions.

1. Draw a control flow graph instruction-by-instruction. (4')

2. Fill up the following def/use/in/out chart. (8')

instr	def	use	in	out
t1 <- r1				
t2 <- r2				
i <- r3				
j <- 1				

jmp L3				
r3,r4 <- i / j				
cmp r4, 0				
jne L2				
r3 <- j				
call print				
j <- j + 1				
cmp j, i				
jl L1				
r2 <- t2				
r1 <- t1				
ret				

3. Draw the interference graph for the program. Please use dashed lines for move edges and solid line for real interference edges. (4')

4. The heuristic to decide which temporary to spill is the same as that in the Tiger book. i.e. The spill priority is calculated as:

```
Spill Priority =
(Uses+Defs-outside-loop + Uses+Defs-within-loop * 10) / Degree
```

Adopt the graph-coloring algorithm to allocate registers for temporaries. Write down the instructions after register allocation. (16')

NOTE: You must write down **every decision you make** such as simplifying, spilling, coalescing. And you also need to write down **new instructions** and draw **new interference graph after each spilling**. If you answer is too simple, you will get a part of score.