上 海 交 通 大 学 试 卷（A 卷）

（ 2019 至 2020 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: Type Checking (16 points)

1.

2.

Problem 2: Static Link and Escape Calculation (30 points)

1.

我承诺，我将严格遵守考试纪律。

承诺人：_____

题号	1	2	3	4					
得分									
批阅人(流水阅卷教师签名处)									

2.

3.

Variable	**Escape(Y/N)**	**Your Reason**
maxN (Line 4)		
emptySymbol(Line 5)		
queenSymbol(Line 9)		
col(Line 17)		

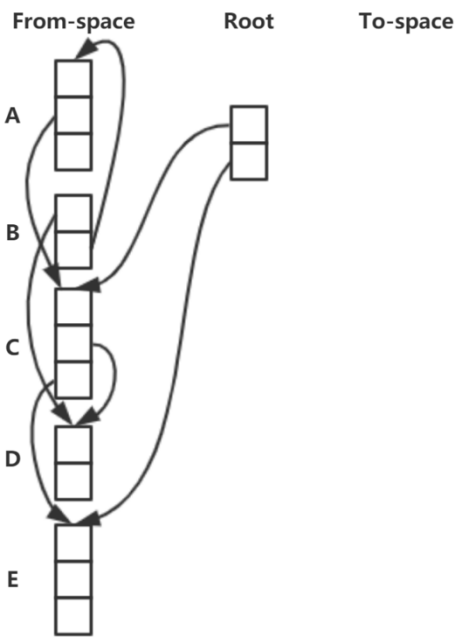
Problem 3: Garbage Collection (20 points)

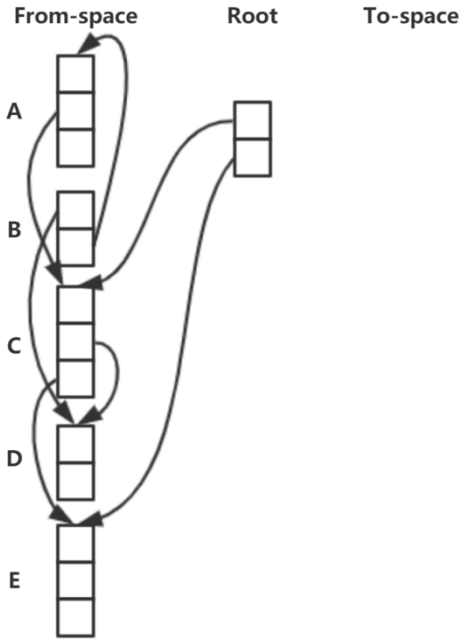
1.

2.

3.

4.





Problem 4: Register allocation (34 points)

1.

2.

instr	def	use	in	out
t1 <- r3				
t2 <- r4				
x <- r1				
y <- r2				
r1 <- y				
call Q				
tmp <- r0				
r1 <- x				
call Q				
add tmp, r0				
r3 <- t1				
r4 <- t2				
ret				

3.

4.

Problem 1: Type Checking (16 points)

```
1   function fibonacci(n: int) = (
2     let
3       type number = int
4
5       var output := "Fibonacci series: "
6       var t1 : number := 0
7       var t2 : number := 1
8       var nextTerm := t1 + t2
9
10      function next() = (
11        let
12          var tmp := 0
13        in
14          t1 := t2;
15          t2 := nextTerm;
16          tmp := t1 + t2;
17          nextTerm := tmp
18        end
19      )
20    in
21      print(output);
22      for i := 1 to n
23      do (
24        let
25          var output := t1
26        in
27          printi(output);
28          print(", ")
29        end;
30
31        next()
32      );
33      print("\n")
34    end
35  )
```

In tiger compiler's semantic analysis phase, we use **symbol tables** (also called **environments**) mapping identifiers to their types. An environment is a set of bindings denoted by the -> arrow. For example, **E**={g -> string, a -> int} means that in environment **E**, the identifier ***a*** is an ***integer*** variable and ***g*** is a ***string*** variable.

1. How many **environments** will be generated in the semantic analysis phase of the function ***fibonacci***? Please write them in detail. (You can answer like this: E0 =

{id1->type1}, E1=E0 + {id2->type2, id3->type3},…) (8’)

2. Each local variable in a program has a **scope** in which it is visible. Through looking into the **active environments**, the compiler can check whether the symbol exists. For the above function ***fibonacci***, you have written the generated **environments** in **Question1**. Please write all **active environments** for the following lines:**1,3-8,14-17,31,33** in ***fibonacci***. (8’)

Problem 2: Static Link and Escape Calculation (30 points)

```
1  /* A program to solve the 1-N queens problem */
2
3  let
4    var maxN := 10
5    var emptySymbol := " ."
6
7    function printSquare(hasQueen:int) =
8       let
9         var queenSymbol := " 0"
10      in print(if hasQueen then queenSymbol else emptySymbol) end
11
12   function nQueens(N:int) =
13     let
14       type intArray = array of int
15
16       var row := intArray [ N ] of 0
17       var col := intArray [ N ] of 0
18       var diag1 := intArray [N+N-1] of 0
19       var diag2 := intArray [N+N-1] of 0
20
21       function printBoard() =
22         (for i := 0 to N-1
23            do(for j := 0 to N-1
24              do printSquare(col[i]=j); print("\n"));
25          print("\n"))
26
27       function try(c:int) =(
28         if c = N then printBoard()
29         else for r := 0 to N-1
30           do if row[r]= 0 & diag1[r+c]=0 & diag2[r-c+N-1]=0
31              then (row[r]:=1; diag1[r+c]:=1; diag2[r-c+N-1]:=1;
32                    col[c]:=r;
33                    try(c+1);
34                    row[r]:=0; diag1[r+c]:=0; diag2[r-c+N-1]:=0)
```

```
35        )
36      in try(0) end
37  in
38    for n := 1 to maxN
39      do(print("Solving ");printi(n);print(" queens problem...\n\n");
40          nQueens(n);
41          print("\n"))
42  end
```

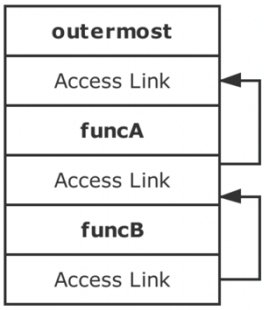
1. Suppose that we implement the nested functions using an access link. Please draw the access link in each of the activation records on the stack based on the following function invocations. (9')

   (1) nQueens->try->try

   (2) nQueens->try->printBoard

   (3) nQueens->try->printBoard->printSquare

   e.g. funcA->funcB



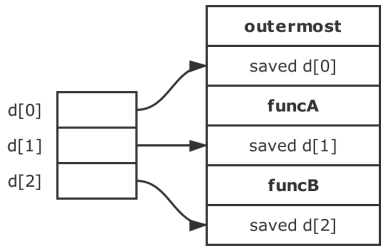
2. Suppose that we implement nested functions using displays. Please draw the display in each of the activation records on the stack based on the following function invocations. (9')

   (1) nQueens->try->try

   (2) nQueens->try->printBoard

   (3) nQueens->try->printBoard->printSquare

   e.g. funcA -> funcB



3. Determine whether the following local variables can escape and explain your reasons. (12′)

Variable	**Escape(Y/N)**	**Your Reason**
maxN (Line 4)		
emptySymbol(Line 5)		
queenSymbol(Line 9)		
col(Line 17)		

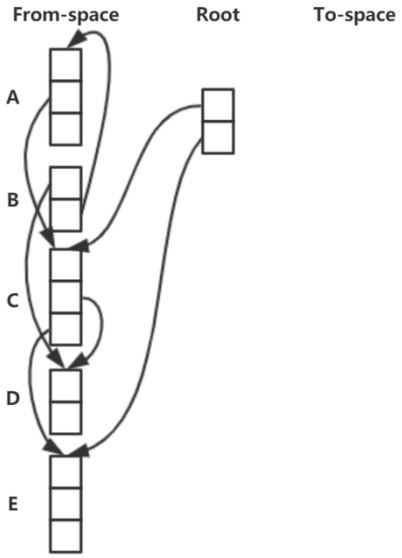
Problem 3: Garbage Collection (20 points)

1. Compare copying collection with mark & sweep collection, what's the **pros & cons** of **copying collection?**(4′)

2. In the course, we have learnt that garbage collection help us recycle heap space occupied by non-live variables. How does the program reuse **space in stack frame(spills in pre-allocated stack) occupied by non-live variables**?(3′)

3. When a garbage collection based on reachability analysis is about to happen, the program runtime usually scan stack frame and use reference variables in stack as roots. **If a reference variable is not live, will we still use it as a root? Why?**(3′)

4. Use copying collection to finish the following datagram. You need to draw the memory state and pointer after **forwarding roots** and **scanning.** You can refer to **figure 13-4 in the Chinese textbook** or **figure 13.10. in the English textbook**. (10′)



Problem 4: Register allocation (34 points)

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

long P(long x, long y)
{
  long u = Q(y);
  long v = Q(x);
  return u + v;
}
	P:
	  t1 <- r3
	  t2 <- r4
	  x <- r1
	  y <- r2
	  r1 <- y
	  call Q
	  tmp <- r0
	  r1 <- x
	  call Q
	  add tmp, r0
	  r3 <- t1
	  r4 <- t2
	  ret

Several things are worth noting in the instructions:

- There are **five** hardware registers in all.
- The compiler passes parameters using registers. (The first parameter goes to **r1**, the second goes to **r2**)
- **r3,r4** is **callee-saved** while **r1,r2** are **caller-saved**.
- The **return value** will be stored in **r0**.
- **t1, t2, x, y, tmp** 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. (10’)

instr	def	use	in	out
t1 <- r3				

t2 <- r4				
x <- r1				
y <- r2				
r1 <- y				
call Q				
tmp <- r0				
r1 <- x				
call Q				
add tmp, r0				
r3 <- t1				
r4 <- t2				
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. 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.