Compiler
2017 Fall Final Examination

Name_____ Student No._____ Score_____

Problem 1: (26 points)

1 只写每次变化的

E0={a->int,b->int}

E1={c->int}

E2={a->string,res->int}

E3={a->int}

或每次可见的均可

E0={a->int,b->int}

E1={ a->int,b->int,c->int}

E2={a->string,b->int,c->int,res->int}

E3={a->int,b->int,c->int} or E3 = E1

其他表达相同意思的均可

2

3-4: E0

8-12: E1

17-18:E2

20:E3

其他表达相同意思的答案均可

3

Alice:

Problem 2: (18 points)

1 (1)

(2)

(3)

2 (1)

(2)

(3)

Problem 3: (26 points)

1.

2

Problem 4: (30 points)

1

2.

instr	def	use	in	out
t1 <- r2				
t2 <- r3				
a <- r1				
res <- 0				
cmp a, 0				

je end				
res <- res * 2				
r1, r2 <- a / 2				
a <- r1				
b <- r2				
res <- res + b				
jmp check				
r1<- res				
r3 <- t2				
r2 <- t1				
return				**r1,r2,r3**

3.

4.

Problem 1: Type Checking (26 points)

```
1   Function gcd (a:int, b:int) =
2   (
3      print_int(a);
4      print_int(b);
5      let
6         var c := 0
7      in
8         while b <> 0
9         do (
10           c := a;
11           a := a - a / b * b;      // i.e., a % b
12           b := c);
13        let
14           var res := b
15           var a := "The result is "
16        in
17           print(a);
18           print_int(res)
19        end
20        print_int(a)
21     end
22  )
```

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 ***gcd*** ? Please write them in detail. (You can answer like this: E0 = {id1→type1},E1={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 ***gcd***, you have written the generated **environments** in **Question1**. Please write all **active environments** for the following lines:**3-4,8-12,17-18,20** in ***gcd***. (You can answer like this: **'3-4:E0,E1**'). (6')

3. Alice and Bob failed to implement the compiler's semantic analysis part. Here are their implementations, please point out why they cannot pass

the ***gcd*** test case, and describe your correct implementation. (12')

- Alice and Bob implemented the symbol table using a single global map<symbol, type>

- In Alice's implementation, when a new identifier is declared, she put the **type of the identifier** into the map using the **symbol of the identifier** as index. When compiling the function ***gcd***, she gets the error message: "Line 20, argument type mismatch"

- In Bob's implementation, when a scope is ended, he deletes all the <symbol, type> pairs declared in this scope from the map. When compiling the function ***gcd***, he gets the error message: "Line 20, undefined symbol 'a'"

Problem 2: Static Link (18 points)

```
type tree = {key: string, left: tree, right: tree}

function prettyprint(tree:tree) : string =
 let
   var output := ""
   function write(s: string) =
      output := concat(output, s)
   function show(n:int, t:tree) =
      let function indent(s:string) =
           (for i := 1 to n
                do write(" ");
           output := concat(output,s);
           write("\n"))
      in
         if t = nil then indent(".")
         else (indent(t.key);
              show(n+1, t.left);
              show(n+1, t.right))
      end
   in
      show(0, tree); output
 end
```

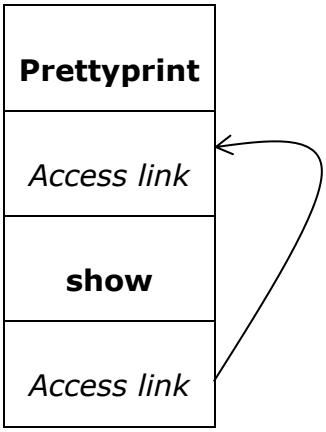
1. Suppose that we implement the nested functions using an access link. Please draw the access link in each of the activation records on the stack based on the following function invocations (9')

(1) prettyprint->show->show

(2) prettyprint->show->indent

(3) prettyprint->show->indent->write

eg: prettyprint->show



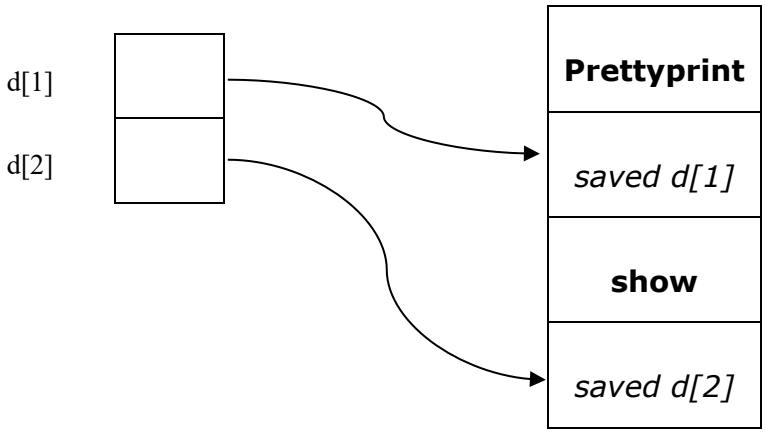
2. Suppose that we implement nested functions using displays. Please draw the display in each of the activation records on the stack based on the following function invocations. (9')

(1) prettyprint->show->show

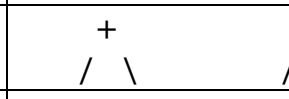
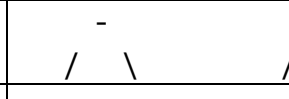
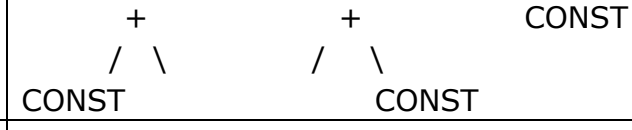
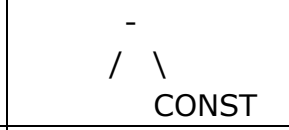
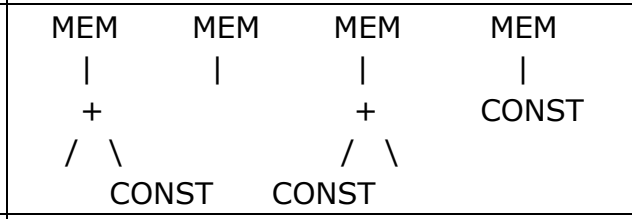
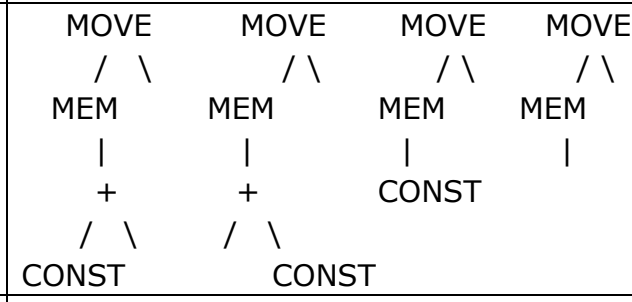
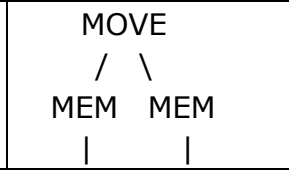
(2) prettyprint->show->indent

(3) prettyprint->show->indent->write

eg: prettyprint->show



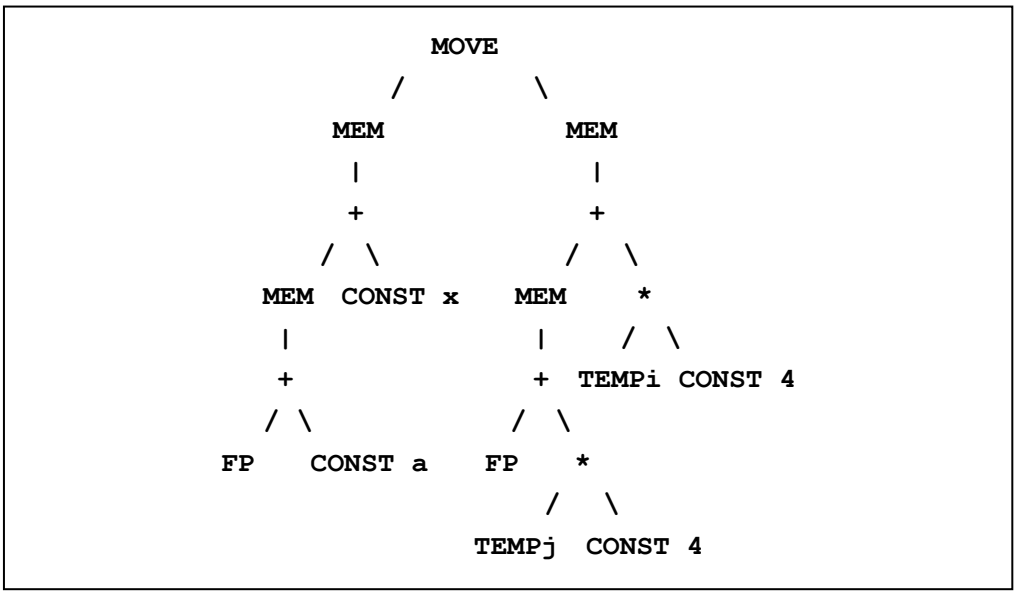
Problem 3: Instruction Selection (26 points)

Name	Effect	Trees	Cost
-	r_i	TEMP	1
ADD MUL	$r_i \leftarrow r_j + r_k$ $r_i \leftarrow r_j * r_k$		1
SUB DIV	$r_i \leftarrow r_j - r_k$ $r_i \leftarrow r_j / r_k$		1
ADDI	$r_i \leftarrow r_j + c$		1
SUBI	$r_i \leftarrow r_j - c$		1
LOAD	$r_i \leftarrow M[r_j + c]$		1
STORE	$M[r_j + c] \leftarrow r_i$		1
MOVEM	$M[r_j] \leftarrow M[r_i]$		2

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.

Consider the following IR tree and answer the questions below.



1. In the course, we have learnt ***maximum 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. (13')

2. ***Maximum munch algorithm*** is an **optimal** tiling algorithm, and will not always generate tiles with smallest cost. A ***dynamic-programming algorithm*** can 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. (13')

Problem 4: Register allocation (30 points)

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

```
int reverse_binary (int a) {
    int res = 0;
    int b;
    while (a != 0) {
        res = res * 2;
        b = a % 2;
        a = a / 2;
        res = res + b;
    }
    return res;
}
```

```
reverse_binary:   t1 <- r2
                  t2 <- r3
                  a <- r1
                  res <- 0
          check:  cmp a, 0
                  je end
          loop:   res <- res * 2
                  r1, r2 <- a / 2
                  a <- r1
                  b <- r2
                  res <- res + b
                  jmp check
          end:    r1<- res
                  r3 <- t2
                  r2 <- t1
                  return
```

Several things are worth noting in the instructions:

- There are **three** hardware registers in all.
- The compiler passes parameters using registers. (The first parameter goes to r1)
- r2, r3 is **callee-saved** while r1 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 **r1** and **r2** respectively.
- The **return value** will be stored in r1.
- t1, t2, a, b, res are temporary registers

Please answer the following questions according to the instructions.

1. Draw a control flow graph instruction-by-instruction. (6’)
2. Fill up the following def/use/in/out chart. (12’)

instr	def	use	in	out
t1 <- r2				
t2 <- r3				
a <- r1				
res <- 0				
cmp a, 0				
je end				
res <- res * 2				

r1, r2 <- a / 2				
a <- r1				
b <- r2				
res <- res + b				
jmp check				
r1<- res				
r3 <- t2				
r2 <- t1				
return				**r1,r2,r3**

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. (8')

NOTE: You will get part of scores if you can provide some intermediate result like spilling and coalescing decisions but fail to write the final instructions.