Compiler
2016 Fall Final Examination

Name________ Student No._________ Score______

Problem 1: (24 points)

1

2

3

Problem 2: (30 points)

1

(a)

(b)

	$	(	)	id	op	P	C	E
0								
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								

(c)

stack	action

2

Problem 3: (18 points)

1.

(1)

(2)

(3)

2.

(1)

(2)

(3)

Problem 4: (28 points)

1

2

instr	def	use	in	out
d <- r3				
a <- r1				
b <- r2				
cmp b,b				
je ret				
c <- a				

r1,r2 <- a/b				
a <- r2				
b <- c				
jmp check				
r1 <- a				
r3 <- d				
return				r1,r3

3

4

Problem 1: Lexical Analysis (24 points)

Suppose we want to specify some lexical tokens of a programming language.

```
/*lex definition*/
Integer                      0 |
                             [1-9] (Digits)?
Digits                       Digit+
Digit                        [0-9]
%%
/*regular expressions and actions*/
if | else | while | for | goto  {print("1"); return KEYWORD;}
[a-zA-Z_][a-zA-Z_0-9]*          {print("2"); return ID;}
Integer                         {print("3"); return INTEGER;}
FloatPoint                      {print("4"); return FLOATPOINT;}
(" " | "\n")                    {print("5"); }
.                               {error(); }
```

NOTE: (**a**)? means 0 or 1 **a**.

1. Please finish the regular expression of FloatPoint. The requirements are below:

   (1) A floating-point has the following parts: an integer part, a decimal point, a fraction part, an exponent and a type suffix.

   (2) There should be at least one digit in either the integer part or the fraction part.

   (3) A floating-point should contain at least one part from those three: a decimal point, an exponent, or a float type suffix.

   (4) All other parts are optional.

   (5) The exponent, if present, is indicated by the ASCII letter e or E followed by an optionally signed integer.

   (6) The type suffix can be f or F(for float), d or D(for double). (8')

   Examples of FloatPoint:

8.7 4. .6 3.2e10 17e-2 0.2e+10d 78f 23e-4D

001. .200 0.e+4 .5e-0d

These are not FloatPoint:

4 . 4e++2 3.2ff 5.1fe-3 2.9e

2. Draw the minimized DFA that accepts on FloatPoint or Integer. For any regular expression, the minimized DFA is a unique DFA having the smallest number of states that accepts it. You will get part of scores if your DFA is not minimized. (Note that accepting state with different identities **CAN'T** be merged) (12')

3. What will the output be on the following input? Assume there is nothing to the right of the last visible character on each line (including the last line) except a newline character **\n** and that the following input is to be scanned all at once: (4')

```
0if  else_.3for

while3.00e-7_6a

000e  goto
```

Problem 2: Grammar (30 points)

The following is a grammar for an abstracted Lisp-like language, in which every expression is represented as a list containing an operator and any number of operands, which can in turn be expressions or identifiers.

NOTE: The start symbol of the grammar is **P**.

```
P    —> ( op C          (1)
C    —> ) |          (2)
         E C            (3)
E    —> ( op C |        (4)
        id              (5)
```

1. Construct the state graph (DFA) for this grammar using LR(1) items. (12')

2. Follow the LR(1) procedure to construct the parsing table for the above DFA. (10')

3. Show the operation of such a parser on the input string "(op id (op) (op id))". (You can merge multiple continual shifts or reduces into one. E.g. for three shifts, write as "S 3"; for reduce 1, reduce 2, reduce 3, write as "R 123") (8')

Problem 3: Static Link (18points)

Please answer the following questions according to the definition of nested functions below.

```
function primeProg():int =
let
    var N := 200
    type intArray = array of int
    var prime_flag := intArray [ N ] of 0
    function printPrime(value:int) =
       (print("value:");printi(value);print("\n"))
    function primeGen(num:int):int =
    (let
        function isPrime(value:int):int =
       let var ret := 1
         in
           (for i := 2 to value-1
              do if value % i = 0 then (ret:=0;break);
             if ret=1 then printPrime(value);ret)
         end
     in
       (if num < N-1 then primeGen(num+1);
        prime_flag[num]:= isPrime(num))
     end)
 in primeGen(2)
end
```

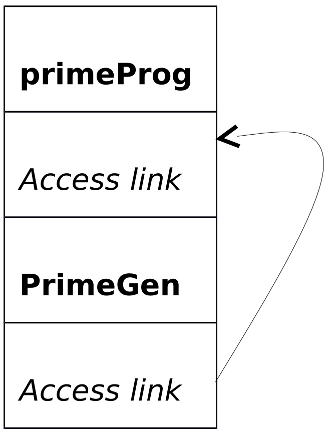
1. Suppose that we implement the nested functions using an access link. Please draw the access link in each of the activation records on the stack based on the following function invocations (9')

   (1) primeProg->primeGen->primeGen

   (2) primeProg->primeGen->primeGen->isPrime

   (3) primeProg->primeGen->primeGen->isPrime->printPrime

   eg: primeProg->primeGen



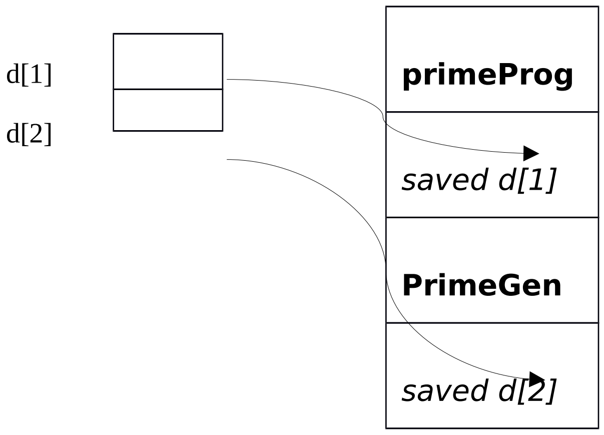
2. Suppose that we implement nested functions using displays. Please draw the display in each of the activation records on the stack based on the following function invocations. (9’)

   (1) primeProg->primeGen->primeGen

   (2) primeProg->primeGen->primeGen->isPrime

   (3) primeProg->primeGen->primeGen->isPrime->printPrime

   eg: primeProg->primeGen



Problem 4: Register allocation (28 points)

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

```
int gcd(int a, int b){             gcd: d <- r3
   while (b != 0) {                     a <- r1
      int c = a;                        b <- r2
      a = a % b;                 check: cmp b, 0
      b = c;                            je end
   }                              loop: c <- a
   return b;                            r1,r2 <- a / b
}                                       a <- r2
                                        b <- c
                                        jmp check
                                   end: r1 <- a
                                        r3 <- d
                                        return
```

Several things are worth noting in the instructions:
(1). There are three hardware parameters in all.
(2). It passes parameters using registers. (The first parameter goes to r1 while the second one goes to r2)
(3). r3 is callee-saved while r1 and r2 are caller-saved.
(4). 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.
(5). The return value will be stored in r1.
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. (10')

instr	def	use	in	out
d <- r3				
a <- r1				
b <- r2				
cmp b,0				
je end				
c <- a				
r1,r2 <- a/b				
a <- r2				
b <- c				
jmp check				

r1 <- a				
r3 <- d				
return				**r1,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
```

Try to 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.