Second Midterm Exam
CS 314, Spring '23
April 14
MIDTERM B - Sample Solution

DO NOT OPEN THE EXAM UNTIL YOU ARE TOLD TO DO SO

Name: ______________________________

Rutgers ID number: ______________________________

Section: _____

WRITE YOUR NAME ON EACH PAGE IN THE UPPER RIGHT CORNER.

Instructions

We have tried to provide enough information to allow you to answer each of the questions. If you need additional information, make a *reasonable* assumption, write down the assumption with your answer, and answer the question. There are **5** problems, and the exam has **6** pages. Make sure that you have all pages. The exam is worth **300** points. You have **70 minutes** to answer the questions. Good luck!

This table is for grading purposes only

1	/ 100
2	/ 90
3	/ 50
4	/ 30
5	/ 30
total	/ 300

NAME: ________________

Problem 1 - Scoping (100 pts)

Assume the following program while answering the questions.

```
Program A()
 {    x, y: integer;

      procedure B()
       {  z: integer;
          z = 3;
          x = y + z; // <---  (*1*)
       }

      procedure C()
       {  y: integer;
          y = 3;
          call B();
       }

      // statement body of A
      x = 5; y = 6;
      call C();
      print x, y;
 }
```

ILOC Instructions

instr. format	description	semantics
	memory instructions	
`loadI` <const> $\Rightarrow r_x$	load constant value <const> into register r_x	$r_x \leftarrow$ <const>
`loadAI` r_x , <const> $\Rightarrow r_y$	load value of MEM(r_x + <const>) into r_y	$r_y \leftarrow$ MEM(r_x + <const>)
`storeAI` $r_x \Rightarrow r_y$, <const>	store value in r_x into MEM(r_y + <const>)	MEM(r_y + <const>) $\leftarrow r_x$
	arithmetic instructions	
`add` r_x , $r_y \Rightarrow r_z$	add contents of registers r_x and r_y , and store result into register r_z	$r_z \leftarrow r_x + r_y$
`sub` r_x , $r_y \Rightarrow r_z$	subtract contents of register r_x from register r_y , and store result into register r_z	$r_z \leftarrow r_x - r_y$

1. Show the output of a program execution (20 pts)

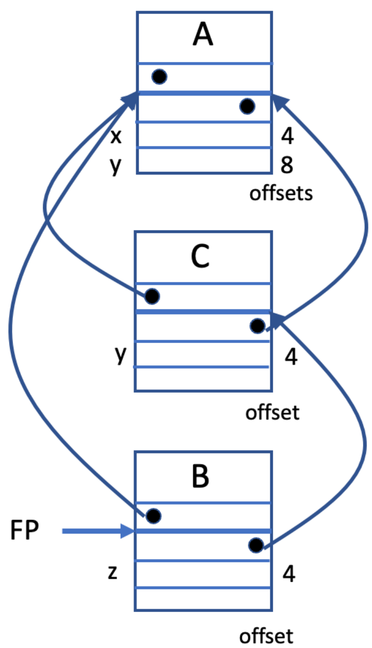
   x= 9 , y= 6 assuming **static (lexical) scoping**

   x= 6 , y= 6 assuming **dynamic scoping**

NAME: ______________

2. Show the runtime stack when execution reaches the point marked (*1*) in the code. Each stack frame has to be labeled with its corresponding procedure name, and has to contain the names of locally allocated variables. **DO NOT** include the values of the variables. **DO** include all control and access links between stack frames. **Use the stack frame layout as discussed in class and in the homework**, with arrows for pointer values to specific memory locations in the frames. **For variables, explicitly specify their positive offsets**. (50 pts)

```
          ^      ----------------
           \    |     . . .      |
            \   |----------------|      ^
             --|- access link   |     /
FP ---->        |----------------|    /
                |  control link -|----
                |----------------|
                |     local      |
                |   variables    |
                 ----------------
```



3. Show the basic (non-optimized) ILOC code generated for statement (∗1∗) in procedure `B` assuming **static (lexical) scoping**. You must only use ILOC instructions as listed above. Assume that register $r0$ contains the value of the frame pointer, that integer and pointer values are 4 bytes long, and that all addresses are byte addresses. (30 pts)

loadAI r0, -4 ⇒ r1 ;; access link to level 1 procedure A
loadAI r1, 8 ⇒ r2 ;; value of y in r2
loadAI r0, 4 ⇒ r3 ;; value of z in r3 (z is at level 2)
add r2, r3 ⇒ r4 ;; value of y+z in r4
loadAI r0, -4 ⇒ r5 ;; access link to level 1 procedure A
storeAI r4 ⇒ r5, 4 ;; value of y+z written to x

NAME: _______________

Problem 2 - Optimization Passes of Project 1 (90 pts)

In your first project, you implemented a peep-hole optimizer for ILOC. Assume the following ILOC code

```
 1:   loadI 1024 => r0
 2:   loadI 4 => r1
 3:   loadI 8 => r2
 4:   loadAI r0, 4 => r3
 5:   add r2, r1 => r4
 6:   loadI 12 => r5
 7:   loadI 3 => r6
 8:   mult r5, r6 => r7
 9:   storeAI r7 => r0, 8
10:   outputAI r0, 4
11:   outputAI r0, 8
```

To answer the questions below, you should list the changed instruction (instructions) with its (their) number(s) and how it (they) changed due to the optimization pass. **Do not list instructions that are not changed by the optimization pass**. For example, for dead code elimination, you might answer: `8:  has been deleted`. If no instruciton has been changed, just say `no change`.

1. If you assume a **window size of 3 for constant folding**, show the ILOC code generated by the constant folding pass of the project for the 11 ILOC instructions listed above. Only list the changed instruction(s): (20 pts)

```
8:   loadI 36 => r7
```

2. If you assume a **window size of 3 for strength reduction**, show the ILOC code generated by the strength reduction pass of the project for the 11 ILOC instructions listed above. Only list the changed instruction(s): (20 pts)

```
no change
```

3. Show the ILOC code generated by the **dead code elimination** pass of the project for the 11 ILOC instructions listed above. Only list the changed instruction(s): (30 pts)

```
2:   loadI 4 => r1  -- has been deleted
3:   loadI 8 => r2  -- has been deleted
4:   loadAI r0, 4 => r3  -- has been deleted
5:   add r2, r1 => r4  -- has been deleted
```

4. If you assume a **window size of 4 for constant folding**, show the ILOC code generated by the constant folding pass of the project for the 11 ILOC instructions listed above. Only list the changed instruction(s): (20 pts)

```
5:   loadI 12 => r4
8:   loadI 36 => r7
```

NAME: ________________

Problem 3 - Scheme (50 pts)

Write a function that takes two symbols as input, and replaces all occurrences of the first symbol `x` by the second symbol `y` in the list `l`. You should assume that all inputs are correct, i.e., no error handling is required.

```
(define replace
  (lambda (x y l)
    ...))
...
(replace 'a 'c '(a b c a)) --> (c b c c)
(replace 'c 'd '(a((b)(c d)(((c)))))) --> (a((b)(d d)(((d)))))
(replace 'f 'd '(a((b)(c d)(((e)))))) --> (a((b)(c d)(((e)))))
```

```
(define replace
   (lambda (x y l)
        (cond
           ((null? l) '())
           ((list? (car l)) (cons (replace x y (car l)) (replace x y (cdr l))))
           (else (if (eq? x (car l))
                  (cons y (replace x y (cdr l)))
                  (cons (car l) (replace x y (cdr l))))))))
```

NAME: ________________

Problem 4 – Lambda calculus (30 pts)

Here is an implementation of boolean values in lambda calculus:

true $\equiv$ λa.λb.a *select–first*

false $\equiv$ λa.λb.b *select–second*

not $\equiv$ (λx. ((x true) false))

Prove that the definition of **not** is not correct, i.e., does not work using the representation of `true` and `false` listed above. **You must show every single β-reduction step in your reduction**. Hint: Replace `true` and `false` by their corresponding lambda terms only when necessary for a β-reduction.

In order to show that the given implementation of **not** is incorrect, apply **not** to **true** and see what normal form the β-reduction will produce. If the β-reduction does not produce the representation of **false**, then the definition of **not** is incorrect. Note: Applying **not** to **false** is also possible to show incorrectness.

(not true) $\equiv$ ((λx. ((x true) false)) true)
$\Rightarrow_\beta$ ((true true) false) $\equiv$ (((λa.λb.a) true) false)
$\Rightarrow_\beta$ ((λb. true) false)
$\Rightarrow_\beta$ true

Problem 5 - Parameter Passing (30 pts)

```
program MAIN {
  int a := 3;
  int b := 4;
  procedure X (int z) {
      z := z + a;
      a := 2 * b; }
  /*MAIN*/
  call X(b) ; print(a, b);
}
```

Given the program above, what values does the program print? Please fill in the values of variables `a` and `b` assuming different parameter passing styles (5 pts each).

pass by value	*pass by reference*	*by pass value-result*
a = 8	a = 14	a 8 =
b = 4	b = 7	b = 7