

CMSC430 Spring 2014 Midterm 2

Name _____

Instructions

- You have 75 minutes for to take this exam.
- This exam has a total of 100 points. An average of 45 seconds per point.
- This is a closed book exam. No notes or other aids are allowed.
- If you have a question, please raise your hand and wait for the instructor.
- Answer essay questions concisely using 1-2 sentences. Longer answers are not necessary and a penalty may be applied.
- In order to be eligible for partial credit, show all of your work and clearly indicate your answers.
- Write neatly. Credit cannot be given for illegible answers.

	Problem	Score	Max Score
1	Syntax directed translation & type checking		12
2	Symbol tables		4
3	Intermediate representations		12
4	Code generation		26
5	Control flow analysis		10
6	CFG ordering, dominators, loops		12
7	Dataflow analysis		24
	Total		100

HONOR PLEDGE: I pledge on my honor that
I have not given or received any unauthorized
assistance on this assignment/ examination.

SIGNATURE: _____

1. (12 pts) Syntax directed translation & type checking

Consider the following grammar fragment for an expression for C--:

$$exp \rightarrow \text{CONST} \mid \text{IDENT}_1 \mid \text{IDENT}_2 [exp_1]$$

Assume that the nonterminal exp and terminals IDENT and CONST have an attribute $type$ that can be assigned a range of type values, including $typeBoolean$, $typeInt$, $typeError$, etc. In addition, the terminal IDENT has an attribute $isArray$ that can be true or false. Add syntax-directed type checking rules to the production $exp \rightarrow \text{IDENT}_2 [exp_1]$ to enforce the following:

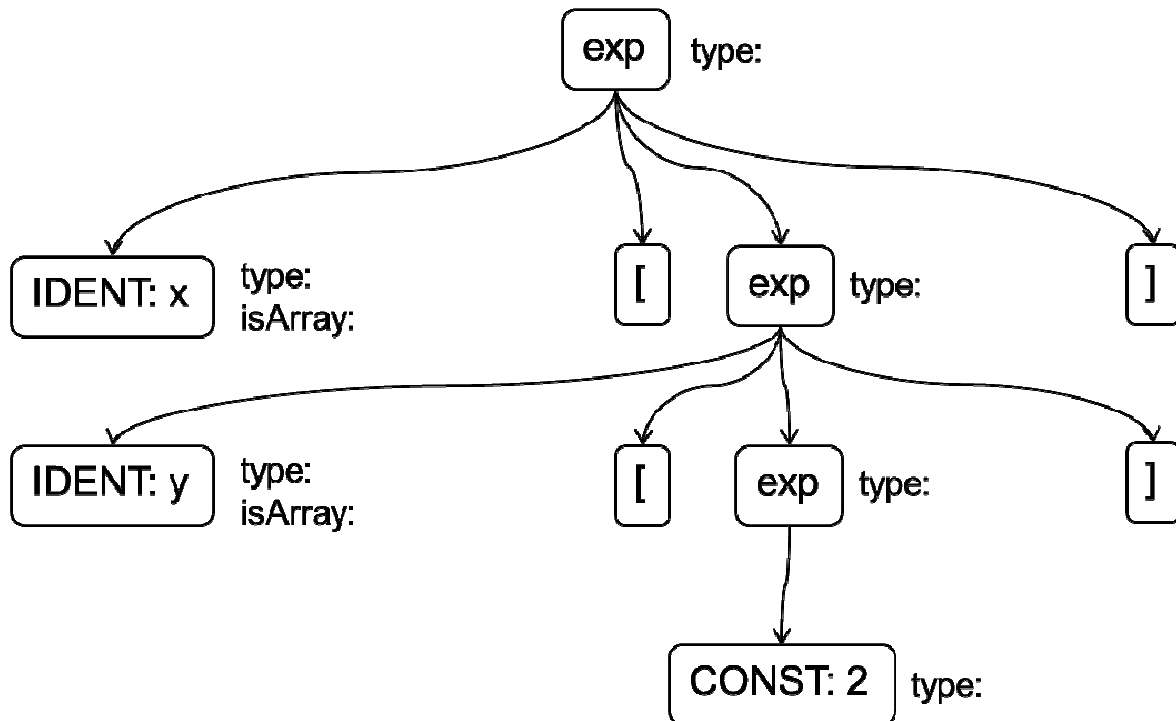
- a. (2 pts) IDENT_2 must be an array. Otherwise call `parser.msg("Misuse of array")`.
- b. (2 pts) The subscript expression exp_1 must be $typeInt$. Otherwise call `parser.msg("Integer required")`.
- c. (2 pts) The expression type is assigned the type of IDENT_2 if both (a) and (b) are true, else it is assigned $typeError$.

Answer:

$$\begin{array}{l} exp \rightarrow \text{IDENT}_2 [exp_1] \\ \{ : \end{array}$$

:}

- d. (6 pts) Given the parse tree for $x[y[2]]$, where x is a BOOL array, y is an INT array, and 2 is an INT, show how syntax directed translation can calculate the type of the expression using your type checking rules by annotating the parse tree. Show the value calculated for each attribute, and draw arrows linking the attributes used in the calculation.



2. (4 pts) Symbol tables

Show the state of lexically nested symbol tables when the compiler reaches the point marked HERE in the following code fragment:

```

int a,b;
void foo( ) { int b; { int c;} }
void bar( ) { int a, { int d,e;} /* HERE */ }
  
```

3. (12 pts) Intermediate Representations

Translate the following arithmetic expression: $x = b - (a * a)$

(2 pts) abstract syntax tree (AST)	(2 pts) directed acyclic graph (DAG)
(4 pts) 3-address code	(4 pts) Java stack code

Java Stack Code	Effect
nop	none
ldc_int c	push constant <i>c</i> onto stack
iload index(x)	push local variable <i>X</i> onto stack
istore index(x)	pop stack, store in local variable <i>X</i>
iadd	pop 2 elems off stack, add, push
isub	pop 2 elems, subtract top from 2nd, push
imult	pop 2 elems off stack, multiply, push
ineg	pop stack, negate, push
goto L	jump to handle <i>L</i>
ifeq L	pop stack, jump to handle <i>L</i> if zero
ifcmpeq L	pop 2 elems, jump to <i>L</i> if equal
ifcmpgt L	pop 2 elems, jump to <i>L</i> if 1st greater
dup	duplicate top of stack
pop	pop top of stack
swap	swap top two positions of stack

3-addr Instruction	Effect
load R1 x	$R1 \leftarrow x$
store x R1	$x \leftarrow R1$
add R1 R2 R3	$R1 \leftarrow R2 + R3$
sub R1 R2 R3	$R1 \leftarrow R2 - R3$
mult R1 R2 R3	$R1 \leftarrow R2 * R3$

Attribute	Holds
AstNode.code	list of instructions
Function	Effect
genInst(<i>X</i>)	create new instruction <i>X</i> returns handle to instruction
append(...)	concatenates lists of instructions

if stack is ... X Y (Y is at top of stack) then if_icmpeq L → pop X, Y from stack, jump to L if X = Y if_icmpgt L → pop X, Y from stack, jump to L if X > Y if_icmplt L → pop X, Y from stack, jump to L if X < Y
--

3.step(2,2) { puts "x" } → "" 3.step(3,2) { puts "x" } → "x" 3.step(4,2) { puts "x" } → "xx" 3.step(5,2) { puts "x" } → "xxx" 3.step(6,2) { puts "x" } → "xxxx"

4. (26 pts) Code generation.

Your compiler generates Java byte codes using attributes (code) and helper functions (genInst, append) found in the table on previous page. Add support in your compiler for a Ruby style STEP loop, where $x.step(y,z) \{ \dots \}$ will iterate with *index* starting at x , incrementing *index* by z each iteration, and repeating while $index \leq y$. Note the loop will not be executed if $x > y$. Write syntax-directed actions needed to generate code for a Ruby style STEP loop:

$$stmt \rightarrow exp_1 . STEP (exp_2 , exp_3) \{ stmtList \}$$
$$\{ : stmt.code = ?? : \}$$

5. (10 pts) Control flow analysis

Consider the following code:

```
<I1>      a := b
<I2>  L1:  b := c
<I3>  L2:  if (...) goto L4
<I4>      c := b
<I5>  L3:  d := a
<I6>  L4:  goto L1
<I7>  L5:  b := a
<I8>  L6:  if (...) goto L2
<I9>      c := a
```

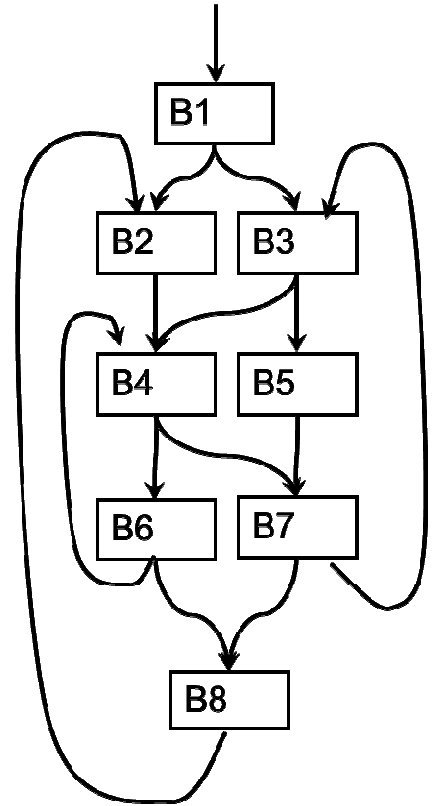
a. (8 pts) Find basic blocks and draw the control flow graph (CFG).

b. (2 pts) What is the compiler term used to describe statement I7, I8, and I9?

6. (12 pts) CFG ordering, dominators, loops

Consider the following control flow graph (CFG), where B1 is the start of the program.

a. (2 pts) List all basic blocks dominated by B3.



b. (2 pts) List all basic blocks dominated by B4.

c. (4 pts) List all the loops in the control flow graph. For each loop list all basic blocks in the loop, as well any back edges in the loop.

d. (4 pts) Give TWO reverse Postorder numberings of the CFG for Problem 7 on the next page.

7. (24 pts) Dataflow analysis

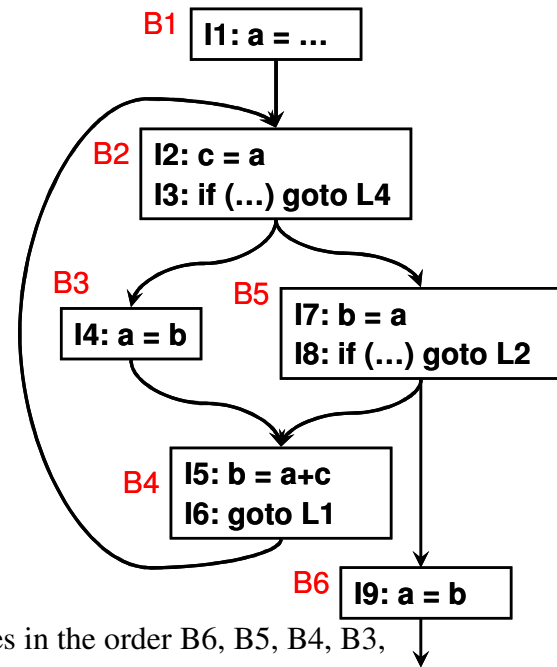
Calculate Live Variables for the following code:

```

<I1>      a := ...
<I2>  L1:  c := a
<I3>      if (...) goto L4
<I4>      a := b
<I5>  L2:  b := a+c
<I6>      goto L1
<I7>  L4:  b := a
<I8>      if (...) goto L2
<I9>      a := b
    
```

a. (8 pts) Calculate GEN/KILL for each basic block

b. (16 pts) Initializing IN/OUT to $\emptyset$, solve live variables in the order B6, B5, B4, B3, B2, B1, showing IN/OUT for each pass. Stop when a fixed point is reached.



Basic Block	GEN	KILL
B1		
B2		
B3		
B4		
B5		
B6		

Basic Block		Initial	Pass 1	Pass 2	Pass 3	Pass 4
B6	OUT	$\emptyset$				
	IN	$\emptyset$				
B5	OUT	$\emptyset$				
	IN	$\emptyset$				
B4	OUT	$\emptyset$				
	IN	$\emptyset$				
B3	OUT	$\emptyset$				
	IN	$\emptyset$				
B2	OUT	$\emptyset$				
	IN	$\emptyset$				
B1	OUT	$\emptyset$				
	IN	$\emptyset$				