CS3300: Final Exam: Nov 16 2023 (A)

Maximum marks = 90, Time: 3.00 hrs

Name: ______________________________ Roll: ______________

- Write your roll number on every sheet of the answer paper and this QP. You have to submit both.
- **Start the answer to every question in a new page.**
- **Negative marking**. Each incorrect answer for a True/False question will lead to deduction of 0.5 mark.
- You may make any reasonable assumptions that you think are necessary; but state them clearly.
- Total questions=9. Total marks=90.

---

1. **Lambdas, Type-casts, InstanceOf Checks**

   (a) Consider the following code in a Java like language that supports lambdas with no arguments. The syntax of lambdas is similar to that of Java with some minor simplifications.

```
class A{
 int f1;
 void foo(){
  f1 = 2;
  int a = 2, b = 5;
  y = lambda () -> {
    int b = f1;
    System.out.println(a+b);
  }
  ... Additional code not shown ...
 } ... Additional code not shown ...
}
```

   Show the contents of the closure to be generated by the compiler for the given lambda. [4]

   (b) Consider the following Java code.

```
class A { ... }
class B extends A { ... }
class C{
 void foo(boolean flag){
  A x;
  if (flag) x = new B();
  else x = new A();
  B y = (B) x;
  ...
 } }
```

   (i) Show the code that has to be generated for the type-cast statement. [2]
   (ii) Also clearly show the additional fields to be added to the objects allocated to support type-cast operations. [2]

   (c) In Java exception thrown in one function may be caught in a different function. True/False. [1 mark]

   (d) Consider the code shown below:

```
for (int i=0;i<100;++i){ throw new Exception(); }
```

   . The number of exceptions thrown by the above code = ——— [1 mark]

2. **Dimensions of Analysis and Constant Propagation**

   (a) Consider the following code.

   ```
   main(){
     int a = 2;
     int b = 3;
     int c = a + b;
     b = b + 1;
     L1:
   }
   ```

   Which of the variables would be considered as constants, using flow-insensitive analysis and which of the variables will be considered as constants using flow-sensitive analysis at `L1`? [2+2]

   (b) Consider the following code.

   ```
   main(){
     int a = foo(4, 4);
     int b = foo(3, 5);
   }
   int foo(int x, int y){
     int t = (x+y)/2;
     return t;
   }
   ```

   Using context insensitive analysis, which of the variables `a`, `b`, `t` will be considered constants? [2 marks]
   Using context sensitive analysis, which of the variables `a`, `b` will be considered constants? [2 marks]

   (c) If no function is called more than once, then context-sensitive and context-insensitive analysis will produce the same results. True/False [1 mark]

   (d) Constant propagation can help reduce code size. True/False. [1 mark]

3. **Loop Analysis**

   (a) Consider the following code:

   ```
   for (i=0;i<n;++i){ S }
   ```

   Write the code after unrolling the body by a factor of 4. Hint: `n` can be any arbitrary integer.

   (b) Consider the following code:

   ```
   S1: x = 2
   S2: y = x + 3;
   S3: x = y * x;
   S4: z = y;
   ```

   Choose, which of input, output, anti and flow-dependencies exist between `S1` and `S2`, `S1` and `S3`, `S2` and `S3`, and `S1` and `S4`. If no dependency exists, you can write "no-dependency".

   (c) Loop-invariants factoring reduces code size. True/False

   (d) Overly aggressive loop unrolling can worsen the execution time performance. True/False.

4. **Optimizations in Basic Blocks**

   (a) Consider the following code.

   ```
   x = 1
   y = 2
   p = x + y
   z = x + y + 2
   x = z * x
   y = x + x
   ```

   Draw the DAG for the above code. [4 marks]

(b) Consider the following pair of instructions.

```
...           // Some code not shown
LD R1 [M1] // loads from the designated memory location M1 to register R1
ST [M1] R1 // stores to the designated memory location M1 from register R1
...           // Some code not shown
```

State the conditions under which (i) both the statements can be removed. (ii) only the second instruction can be removed. [2 + 2 marks]

(c) During algebraic simplifications,

i. $x * 2$ can be replaced by: ———- [0.5 mark]

ii. $x/5.0$ can be replaced by: ———- [0.5 mark]

(d) An expression $0/x$ can always be replaced by 0, using algebraic simplification. True/False [1 mark]

5. **Liveness and Register Allocation**

(a) Consider the following code.

```
L0 : if (x2 <= 0) then goto L2
L1 : goto L8
L2 : p = x1 / x2
L3 : q = p * x2
L4 : r = x1 - q
L5 : x1 = x2
L6 : x2 = r
L7 : goto L0
L8 : o = x1
L9 : return o;
```

Give the live-ranges (as sets of instructions) of all the variables used in the above code. [4 marks] Draw the interference graph. [2 marks] Use Kempe's heuristic to assign registers, assuming the availability of two registers. Show how many variables (and which ones) will be spilled? [2 marks]

(b) Give the flow equations to compute the IN and OUT for liveness analysis. [1 mark]

(c) A variable live out at a node must be live in at all of its successors. True/False [1 mark]

6. **Basic Blocks and Control Flow Analysis**

(a) Consider the below shown IR of a function to build a CFG.

```
 S1: i = 1
 S2: j = 1
 S3: t1 = i*100
 S4: t2 = t1 + j
 S5: t3 = i + j
 S6: t4 = t3 == 100000
 S7: if t4 return
 S8: a[t2] = t3
 S9: j = j + 1
S10: if (j < 100) goto S4
S11: i = i + 1
S12: if (i < 100) goto S3
S13: return
```

List the index of the leader instructions. [3 marks]. Draw the CFG. [3 marks].

(b) Write an IR code, which leads to a basic block which is both a branch and join node. [2 marks]

(c) Every CFG must have at least two basic blocks. True/False [1 mark]

(d) An Exit node may have multiple predecessors. True/False [1 mark]

7. **IR and IR generation**

   (a) Consider the C code shown below.

```
do {
 x = x + i;
 y = y - i;
 if (x < y) continue;
 i = i + 1;
} while (i < n);
```

   Translate the above code to three address code IR (discussed in the class). [4 marks]

   (b) Consider the following grammar.

```
S -> id = E
E -> E - E
E -> (E)
E -> id
E -> num
```

   Write the SDT to generate three-address IR code. [4 marks]

   (c) Given a production of the form $A \rightarrow X\ Y\ Z$ in an L-attributed grammar, give the evaluation order for the inherited and synthesized attributes of A, X, Y, and Z. [2 mark]

8. (a) Consider the following grammar.

```
T -> T * F
T -> T / F
T -> F
F -> id
```

   Rewrite the grammar by left-factoring and removing left-recursion. [2 + 2 marks]

   (b) Consider the following grammar.

```
S -> aAd | bBd | aBe | bAe
A -> c
B -> c
```

   Construct the LR(1) parsing table for the grammar after constructing the LR(1) item sets [3 + 3 marks].

9. **Potpourri**

   (a) Write a code that will lead to a lexical-error by a C compiler. [2 marks]

   (b) Name the file, which contains the body of the C printf function: —— [1 mark]

   (c) If there is no conflict in the LALR(1) parsing table for a grammar, then the corresponding LR(1) parsing table also will have no conflicts. True/False [1 mark]

   (d) The number of states in LALR(1) CFSM matches that of SLR(1) CFSM. True/False [1 mark]

   (e) Register allocation is a machine dependent optimization. True/False [1 mark]

   (f) Function inlining is a machine independent optimization. True/False [1 mark]

   (g) Peephole optimization is typically a whole-program optimization. True/False [1 mark]

   (h) The caller does not have to store/restore all the caller-save registers. True/False [1 mark]

   (i) A function need not save/restore all the callee-save registers in the prologue/epilogue. [1 mark]