Final Exam (CS3300)

Maximum marks = 90, Time: 3.00 hrs

16-Nov-2017

Read all the instructions and questions carefully. You may make any reasonable assumptions that you think are necessary; but state them clearly.

Leave the first page empty. Start each question on a new page. **For any question (including all its sub-parts), the answer should NOT cross two pages.** The spill over text will be strictly ignored. If you scratch/cross some part of the answer, you may use space from the next page.

1. [18] **Parsing**:

(A) Is the following grammar LL(1)? Is it SLR(1)? Give reasons [6 + 10].

S → A a A b | B b B a   A → ϵ B → ϵ

Multiple choice questions [1+1]:

(B) Which of the following is the right order in terms of the number of states of the CFSM?

(a) LR(1) > LALR(1) > SLR(1)

(b) LR(1) > SLR(1) > LALR(1)

(c) LALR(1) = LR(1) > SLR(1)

(d) None of the above.

(C) Which of the following statements is false?

(a) In C, the presence of semicolons, and comma help in writing conflict free CFGs.

(b) If the grammar is LALR(1), there won't be any shift-reduce conflicts in the generated parsing table.

(c) Unlike LL(1) parsing, LR(1) parsing does not need any stack.

(d) There are grammars that are not LR(1).

2. [18] **IR Generation**:

(A) Consider the following grammar:

```
P → S ; P | ε
S → do {P} while (B) | {S}
S → if (B) S
S → break
S → continue
S → id = E
E → E + E
E → id
B → B < B
B → !(B)
B → id
B → B && B
```

```
x = y;
y = a + b + c;
do {
  if (p) break;
  do {
    if (r < t) {
     if (q) continue;
    }
  } while (q);
} while (!(x < y));
```

Write rules to translate statements generated by the grammar shown to the three-address IR discussed in the slides [12]. Use your stated rules to generate IR for the code shown on the right hand side [4].

Multiple choice questions [1+1]:

(B) Choose the right answer:

(a) Three address codes is a synonym for IR.

(b) Each three-address-code instruction has three operands.

(c) Each high level code must be translated to IR before translating it to assembly.

(d) None of the above.

(C) Choose the right answer:

(a) IR cannot be executed.

(b) Call-by value semantics can be implemented using call-by reference scheme.

(c) No IR has representation for loops.

(d) The number of IRs in a compiler depends on the input program.

3. [18] **General**:

(A) Write a C program to find out that operator && uses short-circuiting; that is, when a conditional such as (a == 2) && (b == 3) is evaluated and the first expression (a == 2) is false, then the second expression (b == 3) is not evaluated. Explain the functioning of your program. [5]

(B) Write the output of the following program, along with the reasoning [5]. The parameter passing convention is mentioned as a comment before each parameter.

```
foo(/*call-by-val*/ int a,
    /*call-by-ref*/ int b,
    /*call-by-val*/ int *c){
    /*call-by-val*/ int *d){
 a = 3;
 b = 4;
 d = malloc(sizeof(int)); *d=5;
 printf("%d %d %d %d\n", a,b,*c,*d);
}
```

```
main(){
  int x = 6;
  int y = 7;
  int *z = &y
  int *p = &x;
  foo(x, y, z, p);
  printf ("%d %d %d %d\n", x, y, *z, *p);
}
```

(C) Draw a parse tree for the expression `3 + 2 + 8 * 4 ^ 6 ^ 7` where `^` is an exponentiation operator. `^` has higher precedence than `*`, which has higher precedence than `+`. Further, `*` and `+` are left associative, while `^` is right associative [6].

Multiple choice questions [1+1]:

(E) Choose the wrong answer:

(a) The precedence of the operators must be same across all the compilers for a given language.

(b) AST cannot have more nodes than the syntax tree.

(c) There is no limit on the number of instructions present in a basic block.

(d) None of the above.

(C) Choose the right answer:

(a) A machine can have infinite number of registers.

(b) In C, the predicate `(a * 0 == 0)` will always succeed.

(c) Missing semicolon is a lexical error.

(d) None of the above.

4. [18] **Control flow analysis**:

(A) Define dominators and post-dominators [2]. Give an algorithm to compute dominators [5] that takes a CFG as input. For the following program, build the CFG [5] and then compute the dominators using your algorithm [4].

```
void bar(){
  La: i=1;
  L0: if (i > n) goto L1;
  Lb: j=1;
  Lc: goto L2;
  L1: j=2;
  L2: k = 3;
  Ld: if (n > 0) goto L3
  Le: L4: p++
  Lf: if (p > 10) return;
  Lg: goto L4
  L3: if (m > 0) return;
  Lh: goto L0
}
```

Multiple choice questions [1+1]:

(B) Choose the right answer:

(a) Dominator relationship is reflexive.

(b) Dominator relationship is commutative.

(c) Post-dominator relationship is commutative.

(d) All of the above.

(C) Choose the wrong answer:

(a) The minimum number of basic-blocks in a CFG=2.

(b) For the three-address-code IR discussed in the slides, if we construct the CFG, any basic-block can have at most two predecessors.

(c) Back edges define only a subset of the loops.

(d) None of the above.

5. [18] **Optimizations**:

Consider the following optimizations: i) Eliminate unreachable code, ii) Fuse jump statements, iii) Eliminate redundant stores, and iv) Loop unrolling. For each of those optimizations [4+4+4+4], (i) show a different snippet of C code (no inline assembly allowed) where you can apply the optimization, ii) show the code after optimization. You need not show the header files, main function etc.

Multiple choice questions [1+1]:

(B) Choose the right option:

(a) The complexity of the optimum register allocation algorithm is $O(N)$, where N is the number of live ranges in the program.

(b) Control must flow from one statement to the one following it in the program.

(c) CFG construction can be used identify dead-code.

(d) None of the above.

(C) Choose the wrong option:

(a) Loop-invariant code motion may help improve the i-cache performance.

(b) Loop unrolling cannot deteriorate the program performance.

(c) Register allocation is a machine specific optimization.

(d) None of the above.