CS3250

THE UNIVERSITY OF WARWICK

Third Year Examinations: Summer 2014

Compiler Design

Time allowed: 3 hours.

Read carefully the instructions on the answer book and make sure that the particulars required are entered on the cover of **each** answer book. Start the answer for each question on a new page. Calculators are neither allowed nor necessary.

Answer **FOUR** questions.

---

1. (a) Textbooks typically describe a compiler as composed of distinct phases linked by unidirectional arrows. Discuss reasons why real compilers are not usually structured in this way. [7]

   (b) Explain clearly the difference between pushdown automata and *deterministic* pushdown automata and the relevance of this distinction to parser construction. [7]

   (c) Consider the grammar below:

   ```
   S -> a S b | A B A | b          A -> a b B A | a | eps
   C -> A A c | d                  B -> A b A | b | C A
   ```

   where `S` is the start symbol and `eps` is the empty string. Calculate the First and Follow sets and the Nullable predicate for each of the nonterminals. [11]

---

2. (a) Consider the following grammar:

   ```
   N -> L            L -> B          B -> "0"
   N -> L "." L      L -> L B        B -> "1"
   ```

   Explain why this is not an LL(1) grammar, and show a grammar for the same language that is LL(1). [8]

   (b) Consider adding multiple assignments of the form

   ```
   var_1 = var_2 = ... var_n = Exp
   ```

   to Java, to give all the variables the value of `Exp`.. Propose a syntax (productions) for this feature such that it can be parsed with an LR(1) parser. Assume that appropriate syntax for `Exp` already exists and all variables have already been declared. [9]

   (c) LR parsing is based on the construction of what is known as a characteristic finite state machine. Explain clearly how the states of the machine are constructed. [8]

Continued

---

3. (a) Explain the difference between synthesized and inherited attributes. [8]

   (b) Consider the following grammar for arithmetic expressions:

```
S -> E              T -> T "*" F          F -> id
E -> E "+" T        T -> F                F -> num
E -> T                                    F -> "(" E ")"
```

   i. Show two specific ways in which error recovery can be added into an LR parser for this language. Illustrate with examples. [8]

   ii. Any arithmetic expression can be associated with its postfix form, which places the operator after the representation of both operands. For example, the postfix of (5+4)*(3+2) is 5 4 + 3 2 + * . Design attribute rules for the above grammar that produce the postfix representation of an expression. [9]

---

4. (a) The use of *abstract machine code* is a popular alternative to three-address code as a class of intermediate representations, as exemplified by the current popularity of the JVM. Compare and contrast these two approaches to intermediate code representation, and explain the benefits of each. [8]

   (b) Explain how the Sethi-Ullman algorithm to minimize the number of registers used works. Consider the statement `a := b * c - 2 * d * (e + 1)`. Translate this statement into generalized assembly code using the algorithm. Assume we have two general-purpose registers available. [8]

   (c) Typically booleans and integers are considered distinct types, but some languages allow integers to be used as booleans, with 0 being equivalent to false and any non-zero value being equivalent to true, while still allowing variables to be declared as booleans. Provide syntex-directed definitions for type checking in such a language. You may make assumptions about any precise restrictions and also assume any functions needed to access symbol table. [9]

Continued

---

5. (a) Describe the main steps that need to be performed when calling and returning from procedures. Explain the difference between caller-saved and callee-saved registers. Why do some architectures support both approaches? [8]

   (b) Class definitions in object-oriented languages are represented via class descriptors. Show the class descriptors generated by compiling the following program:

```
class A           {int x = 2;                  a = new A
                   int f () {...} }            b = new B
class B extends A {int y = 0;                  c = new C
                   int g () {...}}
class C extends B {int g () {...}}
```

   Assume the language does not support multiple inheritance and uses dynamic method lookup. [8]

   (c) Many languages suppport the definition and use of data structures (such as a *struct* or a *record*). Explain (informally but clearly) the type-checking aspects of introducing these user-defined structured types. You need not consider variant records or circular types. [9]

---

6. (a) Explain how data-flow information is generated from three-address code, and illustrate using any one kind of information such as reaching definitions, available expressions, or live variables. [8]

   (b) Many GOTO statements generated by simple compilers are unnecessary. Describe two approaches for optimisation, either local or global, that can reduce the number of unnecessary GOTOs. You may assume any representation for intermediate or object code which will help explain the optimisations. [8]

   (c) Consider the following three-address code snippet:

```
L1:   j = 5
      i = 0
L2:   i = i + 1
      t2 = 4 * j
      t3 = a [t2]
      t4 = 4 * i
      a [t4] = t3
      if i < m goto L2
      return
```

   Name and explain the loop optimisations that can be performed on this code and show the resulting optimised code. [9]

End