THE UNIVERSITY OF WARWICK

Third Year Examinations: Summer 2015

Compiler Design

Time allowed: 2 hours.

Answer **FOUR** questions. Read carefully the instructions on the answer book and make sure that the particulars required are entered on **each** answer book.
Calculators are neither necessary nor allowed.

1. (a) Explain what is meant by a *D-context-free language* and discuss why a context-free grammar is preferred when building a compiler for a language.

   [6]

   (b) Explain what is meant by a *closure algorithm* and what makes this approach useful in programming a compiler.

   [6]

   (c) Consider the following grammar:

   ```
   S -> A $
   A -> B A a          B -> b B c
   A ->  ϵ             B -> A A
   ```

   Calculate Nullable, First, and Follow for A and B. Explain the steps you have taken.

   [13]

2. (a) Describe and contrast the following concepts: top-down parsing, recursive descent parsing, and LL(1) parsing.

   [11]

   (b) Construct an LL parse table for the following grammar, and explain the steps you have taken:

   ```
   S -> A B C          A -> a A | c
   B -> b | ϵ          C -> c
   ```

   [14]

3. (a) Construct an LR(0) state machine for the following grammar:

```
Z -> E $
E -> T                      T -> i
E -> E + T                  T -> ( E )
```

[14]

(b) Most programming languages support several forms of parentheses (curly, round, and square brackets, for example). Considering that many common programming errors have to do with balancing parentheses, explain how compilers could make it easier to debug programs in terms of parentheses balancing errors. Consider both top-down and bottom-up parsers.

[11]

4. (a) Explain and illustrate two different methods to translate Boolean expressions into intermediate code in 3-address form, and compare them in terms of efficiency and speed.

[11]

(b) Explain what is an *activation record*. Describe the steps that need to be performed and code that needs to be generated when translating function declarations and function calls. Illustrate with the following function written in a language with *static binding*, paying special attention to non-local variables:

```
function main () {
     int a, b ;
     read (a) ; read (b);
     print (f (a))  }
function f (x) {
     int y = a + x/2 ;
     if y < b then return (y)
              else return (f (y))  }
```

[14]

Continued

5. (a) Discuss the advantages and disadvantages of compiling to a virtual machine (such as JVM) as opposed to straightforward translation to assembly code.

   [11]

   (b) Consider the following program and function expressed as 3-address code:

```
 x := 11
 y := 13
t1 := x + y
 z := t1 * 2
 a := 17
 b := 19
t2 := a * b
 z := z * t2
```

   Perform the register allocation procedure to assign one of the available 3 registers (R_0, R_1, and R_2) for each variable, and show the resulting code. Include code for spilling if needed. Explain any algorithms and criteria that are used.

   [14]

6. (a) Explain what is meant by *data-flow analysis*. Illustrate using either *reaching definitions* or *live variables* analysis, and explain how the information flows through the following pseudocode:

```
x := a ;
if x = b then y := n else z := n ;
x := x + y ;
```

   [13]

   (b) Explain what is meant by each type of translator below and discuss the usefulness of each approach:

   i. A cross-compiler
   ii. An interpretive compiler
   iii. A bootstrapping compiler

   [12]

End