University of Glasgow

Wednesday, 07 May 2014
9.30 am – 11.00 am
(Duration: 1 hour 30 minutes)

DEGREES OF MSci, MEng, BEng, BSc, MA and MA (Social Sciences)

COMPUTING SCIENCE 3Z: PROGRAMMING LANGUAGES 3

Answer all three questions.

This examination paper is worth a total of 60 marks.

You must not leave the examination room within the first half-hour or the last fifteen minutes of the examination.

1. (*Syntax*)

Box 1 shows parts of the EBNF grammar of the programming language Fun.

Suppose that Fun is to be extended with arrays. All arrays are to be 1-dimensional, and indexed from 0 upwards. The following program illustrates the required extension:

```
# sum(v) returns the sum of all components of v.
func sum (int[] v):
    int s = 0
    int i = 0
    while i < length(v):
        s = s + v[i]
        i = i + 1
    .
    return s
.

# main() reads a year and write the number of days.
proc main ():
    int year = read()
    int[] size = [31,28,31,30,31,30,31,31,30,31,30,31]
    int feb = 1
    if year/4*4 == year:
        size[feb] = size[feb] + 1 .
    write(sum(size))
.
```

A variable `v` of type '`int[]`' is an array of integers. The construct '`v[i]`' uses the value of `i` to index the array `v`. An expression such as '`[31,28,…,31]`' creates an array.

Modify the grammar to allow for the required extension.

[10]

```
        prog  =  var-decl *  proc-decl +  eof
    var-decl  =  type  id  '='  expr
        type  =  'bool'
              |  'int'
         com  =  id  '='  expr
              |  'if'  expr  ':'  seq-com  '.'
              |  …
     seq-com  =  com *
        expr  =  sec-expr ( ( '==' | '<' | '>' )  sec-expr ) ?
    sec-expr  =  prim-expr ( ( '+' | '-' | '*' | '/' )  prim-expr ) *
   prim-expr  =  'false'
              |  'true'
              |  num
              |  id
              |  '('  expr  ')'
              |  …
       …
```

Box 1 Parts of the EBNF grammar of Fun.
(Here *prog* is a program, *var-decl* is a variable declaration, *com* is a command, *seq-com* is a sequential command, *expr* is an expression, *prim-expr* is a primary expression, *id* is an identifier, and *num* is a numeral.)

2. (*Concepts*)

(a) What is meant by the *lifetime* of a variable?

What is the lifetime of:

(i) a global variable?

(ii) a local variable?

(iii) a heap variable?

[6]

(b) Consider the Java program outlined in Box 2. Draw a diagram showing the lifetimes of all global and heap variables created by this program.

[6]

(c) Briefly explain the general concept of *encapsulation* in programming languages. Why is encapsulation an important concept?

[4]

(d) How is encapsulation supported by Java? Illustrate your answer by referring to the Java code of Box 2.

[4]

```
public class Dict {
     // A Dict object is a dictionary.

     // A dictionary is represented by a sorted
     // linked list of words.
     private String word;
     private Dict rest;

     public Dict () { word = null; rest = null; }

     // add(w) adds word w to this dictionary.
     public void add (String w) {…}

     // rem(w) removes word w from this dictionary.
     public void rem (String w) {…}

     public static void main (String[] args) {
          Dict d = new Dict();
          d.add("is");
          d.add("am");
          d.add("are");
          d.rem("is");
     }

}
```

Box 2 Outline of a Java program.

3. (*Implementation*)

(a) Explain the role of the *syntactic analysis*, *contextual analysis*, and *code generation* phases of a compiler. How do these phases communicate with each other?

[3]

(b) Box 3a shows parts of an ANTLR grammar file. Explain in detail what ANTLR does with this grammar file.

[6]

(c) Box 3b shows parts of an ANTLR tree grammar file. Explain in detail what ANTLR does with this tree grammar file.

[6]

(d) Box 3c shows parts of an ANTLR tree grammar file. Explain in detail what ANTLR does with this tree grammar file.

[6]

(e) Suppose that the Fun language is to be extended with an additional assignment command such as the following:

```
s += a * b
```

This command should add the value of '`a*b`' to the value stored in the variable s. The syntax should allow an arbitrary expression to the right of '='.

Show how the files of Boxes 3a, 3b, and 3c should be modified to achieve this extension.

[9]

```
grammar Fun

…

com
     : ID ASSN expr                -> ^(ASSN ID expr)
     | …
     ;

…

ID     : LETTER+ ;
ASSN   : '=' ;
PLUS   : '+' ;
…
```

Box 3a Part of an ANTLR grammar file.

```
tree grammar FunChecker

…

com
     : ^(ASSN ID
          t2=expr)          { lookup ID in the type table,
                                and let its type be t1
                              check that t1 is equivalent to t2
                            }
     | …
     ;

expr                        returns [Type type]
     : ID                   { lookup ID in the type table,
                                and let its type be t
                              set $type to t
                            }
     | ^(PLUS
          t1=expr
          t2=expr)          { check that t1 and t2 are both INT
                              set $type to INT
                            }
     | …
     ;

…
```

Box 3b Part of an ANTLR tree grammar file.
(For clarity, actions are expressed in English rather than Java.)

```
tree grammar FunEncoder

…

com
     : ^(ASSN ID
         expr)              { lookup ID in the address table,
                                 and let its address be d
                              emit the instruction 'STORE d'
                            }
     | …
     ;

expr
     : ID                   { lookup ID in the address table,
                                 and let its address be d
                              emit the instruction 'LOAD d'
                            }
     | ^(PLUS
         expr
         expr)              { emit the instruction 'ADD'
                            }
     | …
     ;

…
```

Box 3c Part of an ANTLR tree grammar file.
(For clarity, actions are expressed in English rather than Java.)