UNIVERSITY OF BIRMINGHAM

School of Computer Science

Third Year – BSc Artificial Intelligence and Computer Science
First Year – UG Affiliate Computer Science/Software Engineering
Third Year – BEng Computer Systems Engineering
Third Year – BSc Computer Science
Third Year – MSci Computer Science
Third Year – MEng Computer Science/Software Engineering
Third Year – BSc Mathematics and Computer Science
Fourth Year – BSc Computer Science with Industrial Year
Third Year – MEng Computer Science/Software Engineering with Industrial Year
Fourth Year – MSci Computer Science with Industrial Year
Fourth Year – BSc Computer Science with Business Management with Industrial Year

06 02578

Compilers & Languages

Summer Examinations 2015

Time allowed: 1 hour 30 minutes

[Answer ALL Questions]

1. Regular expressions are a common way to recognise tokens in an input stream. Regular expressions assign special meanings to, amongst other characters, ".", "*", "?", "[" and "]".

   (a) Write a regular expression which will match real numbers (those with a decimal point and digits following it) such as 3.14159, but will **not** match numbers with a superfluous leading zero such as 02.71828. **[2%]**

   (b) Write a regular expression which will match traditional variable names, which contain letters, digits and underscores, but cannot start with a digit. **[3%]**

   (c) A common notation is "token regexp", where token is a name such as "NUMBER" or "VARIABLE" to be returned to a parser, and "regexp" is a regular expression. Write a set of regular expressions which will unambiguously distinguish between the reals, variables and the standard arithmetic operators ("+", "-", "*", "/") and return suitable token. You can assume that only the first match within a set of regular expressions is taken. **[9%]**

2. Production rules written in BNF notation are used to define the grammar of languages. A grammar *recognises* sentences which are legal in the grammar; a *derivation* is the sequence of productions which produces a legal sentence.

   Consider a C-like language in which the following is a legal program:

```
function f (x) {
  if (x == 1) {
    return 1;
  } else {
    return x * f (x - 1);
  }
}
```

   Assume a tokeniser which returns FUNCTION, OPENBRACKET, CLOSEBRACKET, OPENBRACE, CLOSEBRACE, etc.

   (a) Write a BNF grammar which will recognise functions of this form. Your processing of arithmetic expressions can be limited to simple two-operand expressions. **[10%]**

   (b) Provide a derivation tree which shows how the above fragment is parsed by your grammar. **[10%]**

   (c) The following is not a legal statement in this language:

```
return x * y + + 2;
```

   Draw a partial derivation of this expression which shows where the error is, and explain one strategy by which a parser could attempt to recover from the error and continue to parse subsequent statements. **[6%]**

(d) Recursive descent is one technique for automatically determining the derivation tree for a particular grammar and sentence. Describe the operation of a recursive descent parser, and show how a recursive descent parser would use your grammar to parse the supplied code fragment **[10%]**

3. Optimisation is the process of making machine code run faster than would a direct translation of the source code. Consider the following piece of code in a C- or Java-like language:

```
for (i = 0; i < 3 * 7; i++) {
   j = i * 17.2;
   k = sin (j);
   k = (i * 17.2) + 3.1;
   total += j + k;
}
```

(a) Indicate the optimisations that a compiler could make to improve the performance of this loop without substantially increasing the size of the generated code, naming the techniques that you are using. Assume that multiplication is significantly more expensive than addition. **[10%]**

(b) Code which uses tight loops like that shown above tends to perform badly on processors with large pipelines. Explain what a pipeline is and why tight loops are particularly problematic in the case of machines which make extensive use of pipelining. Explain loop unrolling, and the trade-offs that must be considered when deploying this technique. **[10%]**

4. Most programming languages support the concept of a function, subroutine or method call, in which execution is suspended, some arguments are passed to another piece of code which runs and control then returns to the caller.

   Consider the following code fragment:

```
function f(x, y) {
   integer a;
   integer b;

   a = x + y;
   b = x * y;
   return b - a;
}

function g(void) {
   print f (5, 3) + f (6, 2);
}
```

   (a) Explain how and where the memory for variables "a" and "b" is allocated. **[8%]**

   (b) Explain how the values to be used for parameters "x" and "y" are passed to function f in a C or Java-style "call by value" language. **[8%]**

5. Garbage collection is an important part of modern languages such as Java

   (a) Describe why garbage collection is used. What alternative facilities are provided by languages like C and its associated runtime libraries? **[8%]**

   (b) Garbage collectors are made more complex by the existence of cyclic data structures. Outline one solution which allows them to function in this situation. **[6%]**