Manchester Metropolitan University

2017-2018 Summer Examination Period
Faculty of Science and Engineering
School of Computing, Mathematics & Digital Technology
COMPUTING AND DIGITAL TECHNOLOGY NETWORK
Level 6

6G6Z1110 : PROGRAMMING LANGUAGES: PRINCIPLES AND DESIGN

Duration : 3 hours

Instructions to Candidates

Answer **THREE** questions in total.

Answer the **ONE** and only from **SECTION A** and **TWO** of three from **SECTION B**.

This is an open book exam.

SECTION A – Answer the **ONE** and only question from this section.

1. (a) Write a regular expression that defines an identifier. An identifier, in this context, must start with a lowercase letter, uppercase letter, or underscore, followed by a sequence of zero or more lowercase letters, uppercase letters, underscores, or digits. [5]

(b) Write a regular expression that defines a number. A number, in this context, is either a decimal (i.e. a sequence of one or more digits) or a hexadecimal (i.e. begins with 0x followed by a sequence of characters drawn from [0-9a-fA-F]). [10]

(c) Consider the grammar fragment further below along with the meta-notation table:

⟨foo⟩	means foo is a nonterminal.
foo	(in **bold** font) means that **foo** is a terminal i.e., a token or a part of a token.
[*x*]	means zero or one occurrence of *x*, i.e., *x* is optional; note that brackets in quotes '[' ']' are terminals.
x^*	means zero or more occurrences of x.
x^+,	a comma-separated list of one or more x's.
{ }	large braces are used for grouping; note that braces in quotes '{' '}' are terminals.
|	separates alternatives.

In particular, pay careful attention to the x+, notation: it is not the same as the x+ notation that is commonly used in regular expressions.

⟨method_decl⟩ → {⟨type⟩ | **void**} ⟨id⟩ ([{⟨type⟩ ⟨id⟩}$^+$,]) ⟨block⟩

Provide the ANTLR rule(s) for the above statement. Assume that <block>, <type>, <id> and void are non-terminals and already implemented. If you find it helpful, you can also use the following helper rules to further break down the problem:

⟨method_type⟩ → {⟨type⟩ | **void**}
⟨parameter⟩ → ⟨type⟩ ⟨id⟩

[10]

(d) Explain the method used to enforce precedence in a given grammar. How would you deal with it in a hand-craft complier and how does ANTRL deal with it? [5]

Question 1 continues on the next page

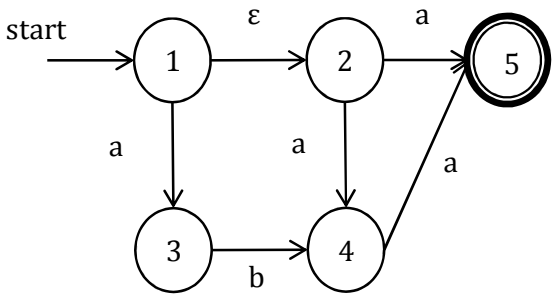
(e) Provide the necessary functions/methods needed to implement a self-contained symbol table. Briefly explain what each method does. [10]

(f) Explain the process of implementing the semantic rule “No identifier is allowed to be used before it is declared”. Assume that you are performing semantic analysis and you are currently looking at an assignment statement that uses a variable ‘a’.

In your answer include also discussion on how the symbol table will be used; which specific “enter” / “exit” methods will be used in the ScopeListener and in what way; and how their “context” will be used. Answer in approximately 150 words. [10]

SECTION B – Answer **TWO** of three questions.

2. (a) Consider the following finite automata:



(i) Give two reasons why this is a non-deterministic finite automata (NFA) and not a deterministic finite automata (DFA). [2]

(ii) Convert the NFA to a DFA. Show your working in the form of a table. [6]

(b) Consider the following grammar:

Expr	→	Expr Op Expr
	|	name
Op	→	+
	|	-
	|	×
	|	÷

(i) Draw two different parse trees that could be constructed for the input a + b × c for this grammar. [4]

(ii) Provide an alternative grammar that ensures that only one tree can be generated for this input, which would result in the correct order of operations upon execution. [2]

Question 2 continues on the next page

(c) Consider the following pseudo code:

```
int main() {
        int a = 5, b = 2, c;
        boolean debug = false;
        int count = get_input();
        if (debug)
                print (b+count);
        for (int i = 0; i < count/2; i++)
                print ("\t");
        print (a*count);
}
```

(i) Identify an optimization that could be made using *code motion*, and write down the optimized code. [1]

(ii) Remove all *dead code* from this code fragment to produce further optimization. [3]

(d) Pipelining is a means to speed up execution of programs, by allowing subcomponents of instructions to be executed in parallel.

(i) Give an example of an instruction dependency that could lead to a stall in the pipeline. [1]

(ii) These types of dependencies can be even more of an issue in superscalar architectures. What strategy is used at the hardware level to reduce the occurrence of such dependencies? [2]

(iii) Very long instruction word architectures depend on compilers to remove such dependencies. Why is this, and what benefit do VLIW architectures have? [4]

3. (a) Consider the following regular expression:

$(00 + 1)^* 1 (0 + 1)$

(i) Which (if any) of the following input strings would be accepted by this regular expression: [2]

1) 0011
2) 00111
3) 011
4) 000
5) 0001

(ii) Construct a non-deterministic finite automata (NFA) that represents this expression. [8]

(b) (i) Left-recursion can be problematic in parsing. What is left-recursion in the context of parsing? [2]

(ii) Why is left-recursion a problem in top-down parsing but not in bottom-up parsing? [2]

(c) Consider the following Java code fragment:

```
public class YourClass {
        int i;
        double d;
        …
}
public class MyClass extends YourClass {
        char c;
        …
        void foo(int val, String s) {
                i = s.length() – val;
                c = s.charAt(i);
        }
}
```

(i) Explain how a symbol table would be constructed to check the types in this code fragment. You either need to show how the symbol table will look like at the point where the last statement in *foo* has been executed or explain the process of how the table has been constructed up to that point. [5]

Question 3 continues on the next page

(ii) Would this code generate a compile-time error? If not, could it generate a run-time error? If yes to either of these questions, explain why. [2]

(d) Consider the following assignment statements, occurring sequentially in a program:

```
a = b+c;
d = a+e;
```

Naively-generated assembly code could look like this:

```
load $0, b          //$0 = b
load $1, c          //$1 = c
add  $0, $0, $1     //$0 = $0 + $1
save $0, a          //a = $0
load $0, a          //$0 = a
load $1, e          //$1 = e
add  $0, $0, $1     //$0 = $0 + $1
save $0, d          //d = $0
```

(i) Which of these instructions should be removed by an optimiser? [1]

(ii) Explain the difference between register allocation and register assignment choices. In what way do they play a role in optimisation? [3]

4. (a) Consider the following grammar:

$$\begin{aligned} A &\rightarrow Ba \\ B &\rightarrow cab \\ &\mid Ab \end{aligned}$$

(i) Explain why this grammar is left-recursive. [4]

(ii) Re-write the grammar to remove the left-recursion. [5]

(b) (i) What are Static type checking and Dynamic type checking? What are the pros and cons of static type checking (as opposed to Dynamic)? [4]

(ii) Explain the concept of structural equivalence for types. Provide an example (in an abstract language if you like) of two types that are structurally equivalent but not name-equivalent. [3]

(c) Why would you perform optimisation in multiple passes, rather than a single pass? [4]

(d) Consider a pipelined processor consisting of five phases:

1. Instruction fetch
2. Register fetch
3. Execute
4. Memory access
5. Register write-back

Assume each of these phases takes one time step.

(i) Consider the following sequence of instructions:

```
add $3, $1, $2
add $5, $3, $4
```

How long would these two instructions take to complete? [2]

(ii) If a third instruction was added after these:

```
sub $6 $2 $1
```

What would be the shortest possible number of steps in which these three operations could execute? Explain why. [3]

END