EXAM PAPERS PRACTICE

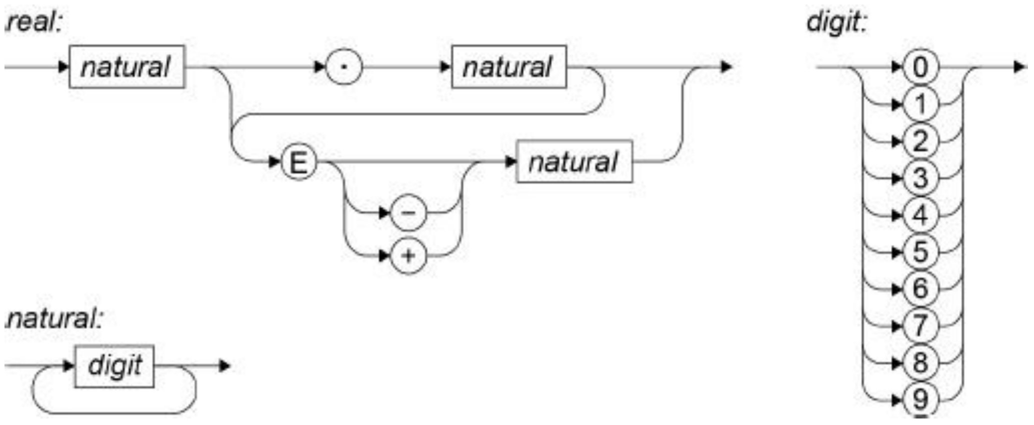
4.3 Context free languages

Name:	
Class:	
Date:	

Time:	**66 minutes**
Marks:	**45 marks**
Comments:	

Q1.

In a particular programming language, the correct syntax for a real number, natural number and digit is defined by the syntax diagrams in the diagram below.



(a) Write **Yes** or **No** in the unshaded cells in the table to identify whether or not the numbers listed in the table are valid real numbers which conform to the correct syntax for this language.

Real number	**Valid? (Yes/No)**
87.000	
97+12	
12.31E+12	

(3)

(b) In Backus-Naur Form (BNF) the following production rule has been written to define a digit:

```
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Write a BNF production rule to define a natural number that is equivalent to the definition in the syntax diagram in the diagram.

____

____

____

____

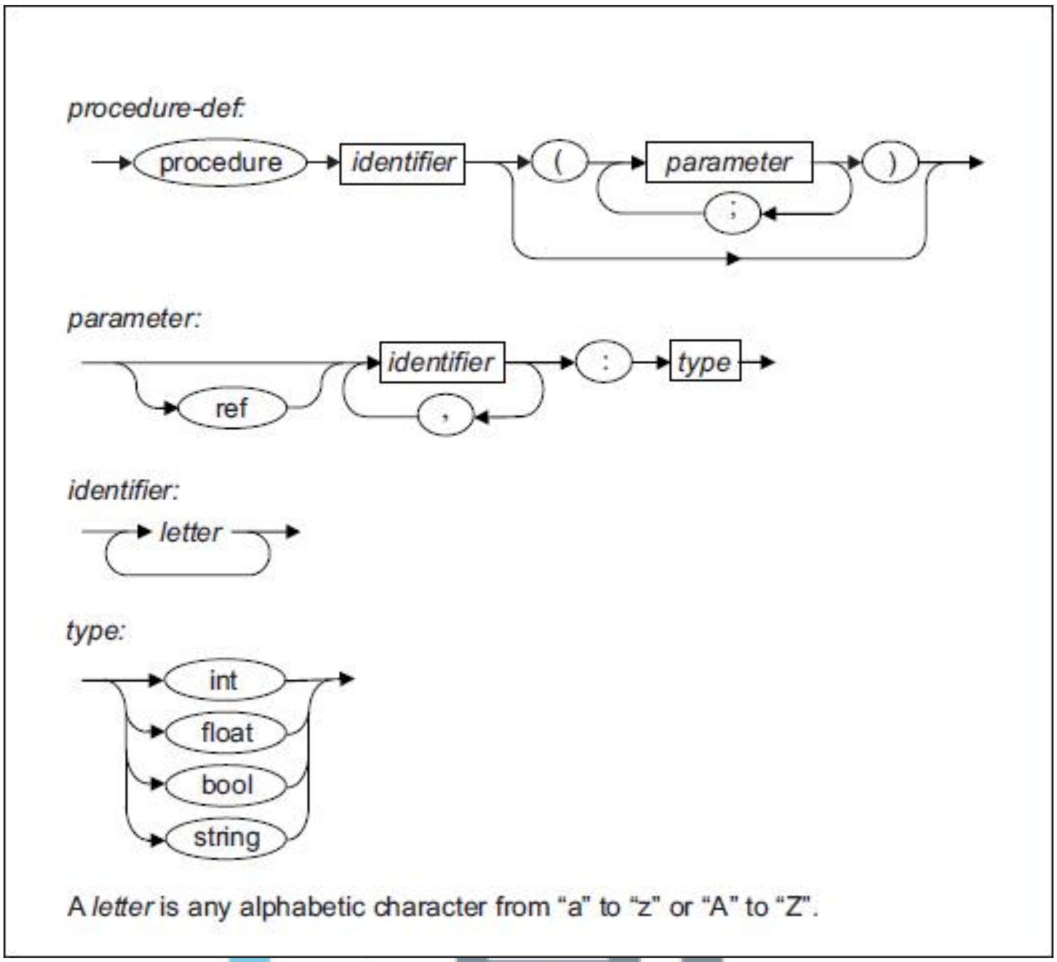
(2)

(Total 5 marks)

Q2.

In a particular programming language, the correct syntax for four different constructs is defined by the syntax diagrams in **Figure 1**.

Figure 1



In this language an example of a valid *identifier* is `loopCount` and an example of a valid *type* is `int`.

(a) For each row in the table below, write **Yes** or **No** in the **Valid?** column to identify whether or not the **Example** is a valid example of the listed **Construct**.

Construct	**Example**	**Valid? (Yes/No)**
identifier	`Game_Over`	
parameter	`ref x,y:bool`	
procedure-def	`procedure square(s:float)`	
procedure-def	`procedure rect(w:int,h:int)`	

(4)

A student has written Backus-Naur Form (BNF) production rules that are supposed to define the same constructs as the syntax diagrams in **Figure 1**. Their BNF rules are shown in **Figure 2**.

Figure 2

```
<procedure-def> ::= procedure <identifier> ( <paramlist> )
<paramlist>     ::= <parameter> | <parameter> ; <paramlist>
<parameter>     ::= <identlist> : <type> |
                    ref <identlist> : <type>
<identlist>     ::= <identifier> | <identifier> , <identlist>
```

```
<identifier>      ::= <letter> | <letter> <identifier>
<type>            ::= int | float | bool
```

A `<letter>` is any alphabetic character from “a” to “z” or “A” to “Z”.

(b) The BNF production rules in **Figure 2** contain two errors. These errors mean that the production rules do not represent the same statement types as the syntax diagrams in **Figure 1**.

Describe the **two** errors.

Error 1: ____

____

____

Error 2: ____

____

____

(2)

(c) The production rule for a `<paramlist>` is recursive.

Explain why recursion has been used in this production rule.

____

____

____

____

(1)

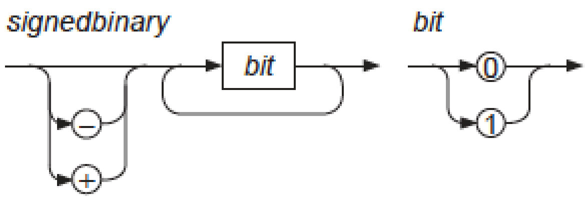
(Total 7 marks)

Q3.

The table shows three definitions of a language for signed binary numbers, each written using a different standard notation.

All three definitions are supposed to be of the same language for signed binary numbers, but one of them contains an error which means that it defines a different language.

1	`<signedbinary> ::= + <binary> | - <binary> | <binary>`
	`<binary> ::= <bit> | <bit> <binary>`
	`<bit> ::= 0 | 1`

2	
3	`(+|-)(0|1)`$^+$

(a) What is the name of the standard notation used in definition **2** in the table?

____

(1)

(b) State the number of the definition (**1 to 3**) in the table that does not define the same language as the other two definitions.

☐

(1)

(c) Explain how the language defined by the definition that you have identified in part **(b)** would differ from the language defined by the other two definitions.

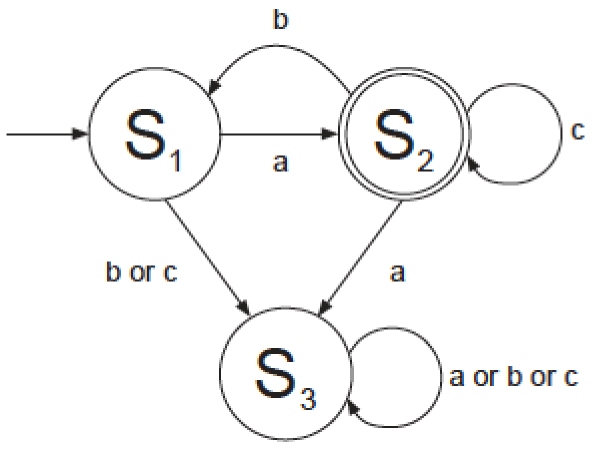
____

____

____

(1)

The diagram shows a finite state automaton that recognises a language.


$$\begin{tikzcd}
\arrow[r] & S_1 \arrow[r, "a"'] \arrow[dr, "b \text{ or } c"'] & S_2 \arrow[l, bend right, "b"'] \arrow[d, "a"] \arrow[loop right, "c"] \\
& & S_3 \arrow[loop right, "a \text{ or } b \text{ or } c"]
\end{tikzcd}$$

(d) Write a regular expression that would recognise the same language as the finite state automaton in the diagram.

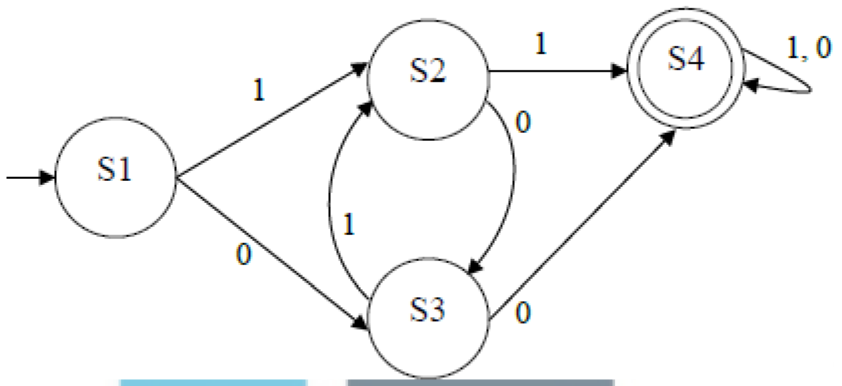
____

(1)
(Total 4 marks)

Q4.

A finite state machine (FSM) can be used to define a language: a string is allowed in a language if it is accepted by the FSM that represents the rules of the language.
Figure 1 shows the state transition diagram for an FSM.

Figure 1



An FSM can be represented as a state transition diagram or as a state transition table. The table below is an incomplete state transition table for **Figure 1** .

(a) Complete the table.

Original state	**Input**	**New state**
S3		
S3		

(1)

(b) Any language that can be defined using an FSM can also be defined using a regular expression.

The FSM in **Figure 1** defines the language that allows all strings containing at least, either two consecutive 1s or two consecutive 0s.

The strings 0110, 00 and 01011 are all accepted by the FSM and so are valid strings in the language.

The strings 1010 and 01 are not accepted by the FSM and so are not valid strings in the language.

Write a regular expression that is equivalent to the FSM shown in **Figure 1**.

____

____

____

____

____

____

(3)

(c) Backus-Naur Form (BNF) can be used to define the rules of a language.

Figure 2 shows an attempt to write a set of BNF production rules to define a language of full names.

Figure 2

Note: underscores (_) have been used to denote spaces.
Note: rule numbers have been included but are not part of the BNF rules.

Rule number

```
1          <fullname> ::= <title>_<name>_<endtitle> |
                          <name> |
                          <title>_<name> |
                          <name>_<endtitle>

2          <title> ::= MRS | MS | MISS | MR | DR | SIR

3          <endtitle> ::= ESQUIRE | OBE | CBE

4          <name> ::= <word> |
                      <name>_<word>

5          <word> ::= <char><word>

6          <char> ::= A | B | C | D | E | F | G | H | I |
                      J | K | L | M | N | O | P | Q | R |
                      S | T | U | V | W | X | Y | Z
```

BNF can be used to define languages that are not possible to define using regular expressions. The language defined in **Figure 2** could not have been defined using regular expressions.

Complete the table below by writing either a **'Y'** for **Yes** or **'N'** for **No** in each row.

Rule number (given in Figure 2)	Could be defined using a regular expression
1	
2	
3	
4	
5	
6	

(1)

(d) There is an error in rule 5 in **Figure 2** which means that no names are defined by the language.

Explain what is wrong with the production rule and rewrite the production rule so that the language does define some names – the names 'BEN D JONES', 'JO GOLOMBEK' and 'ALULIM' should all be defined.

____

____

____

____

(2)
(Total 7 marks)

Q5.

The Backus-Naur Form (BNF) production rules below define the syntax of a number of programming language constructs.

```
<forloop>     ::=  FOR <variable>  = <integer>  TO  <integer>
<variable>    ::=  <letter> | <letter> <string>
<string>      ::=  <character> | <character> <string>
<integer>     ::=  <digit> | <digit> <integer>
<character>   ::=  <digit> | <letter>
<digit>       ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

A `<letter>` is any alphabetic character from `a` to `z` or `A` to `Z`.

(a) The table below contains a list of variable names. Place a tick in each row if the stated variable name is a valid `<variable>` for the production rules above.

You may tick **more than one** box.

<variable>	**Valid? (✓ any number of rows)**
a	
money_paid	
taxrate2	
2ndPlayerName	

(1)

(b) The production rule for an `<integer>` is recursive.

Explain why recursion has been used in this production rule.

____

____

____

(1)

(c) Here is an example of a valid `<forloop>` :

```
FOR count = 1 TO 10
```

The BNF production rules above can be used to check whether or not a `<forloop>` is syntactically correct.

However, it is possible that a programming language statement that is a syntactically correct `<forloop>` may still produce an error when the program that it is part of is compiled.

Describe **one** example of why a syntactically correct `<forloop>` may still produce an error when a program is compiled.

____

____

____

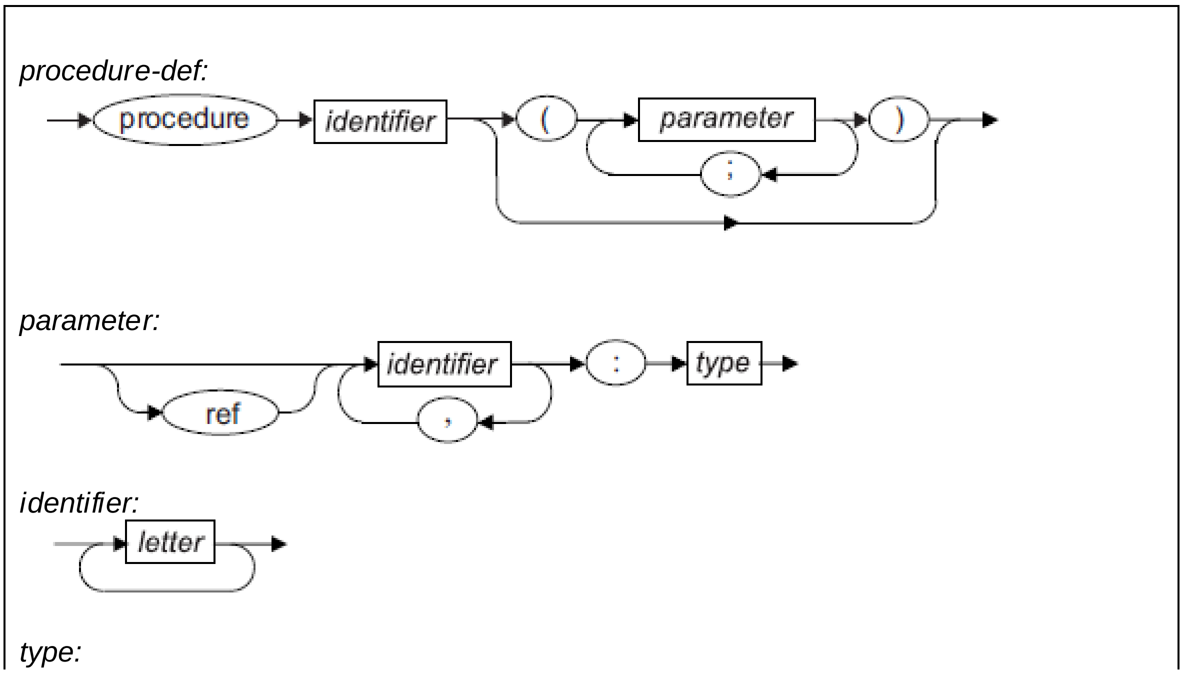
____

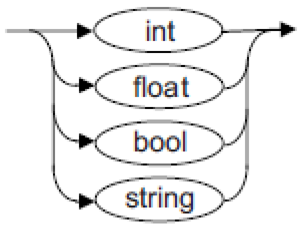
(1)
(Total 3 marks)

Q6.

In a particular programming language, the correct syntax for four different constructs is defined by the syntax diagrams in **Figure 1.**

Figure 1





A *letter* is any alphabetic character from "a" to "z" or "A" to "Z".

In this language an example of a valid *identifier* is `loopCount` and an example of a valid *type* is `int`.

(a) For each row in the table below, write **Yes** or **No** in the empty column to identify whether or not the **Example** is a valid example of the listed **Construct**.

Construct	**Example**	**Valid? (Yes / No)**
identifier	`Player2name`	
parameter	`x,y:bool`	
procedure-def	`procedure square(s:real)`	
procedure-def	`procedure rect(w:int,h:int)`	

(4)

(b) A student has written Backus-Naur Form (BNF) production rules that are supposed to define the same constructs as the syntax diagrams in **Figure 1**. Their BNF rules are shown in **Figure 2**.

Figure 2

```
<procedure-def> ::= procedure <identifier> ( <paramlist> )
<paramlist>     ::= <parameter> | <parameter> ; <paramlist>
<parameter>     ::= <identlist> : <type> |
                    ref <identlist> : <type>
<identlist>     ::= <identifier> | <identifier> , <identlist>
<identifier>    ::= <letter> | <letter> <identifier>
<type>          ::= int | float | bool | char | string
```

A <`letter`> is any alphabetic character from "`a`" to "`z`" or "`A`" to "`Z`".

(i) The BNF production rules in **Figure 2** contain two errors. These errors mean that they do not represent the same statement types as the syntax diagrams in **Figure 1.**

Describe the **two** errors.

Error 1:____

____

Error 2:____

____

(2)

(ii) The production rule for a <`paramlist`> is recursive.

Explain why recursion has been used in this production rule.

____

____

____

____

(1)

(Total 7 marks)

Q7.

The diagram below shows some production rules that have been used to define the syntax of valid mathematical expressions in a particular programming language.

```
<expression> ::= <factor> | <factor> * <factor> | <factor> / <factor>
<factor> ::= <term> | <term> + <term> | <term> - <term>
<term> ::= - <expression> | <number>
<number> ::= <digit> | <digit> <number>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

(a) What notation method has been used in the diagram above?

____

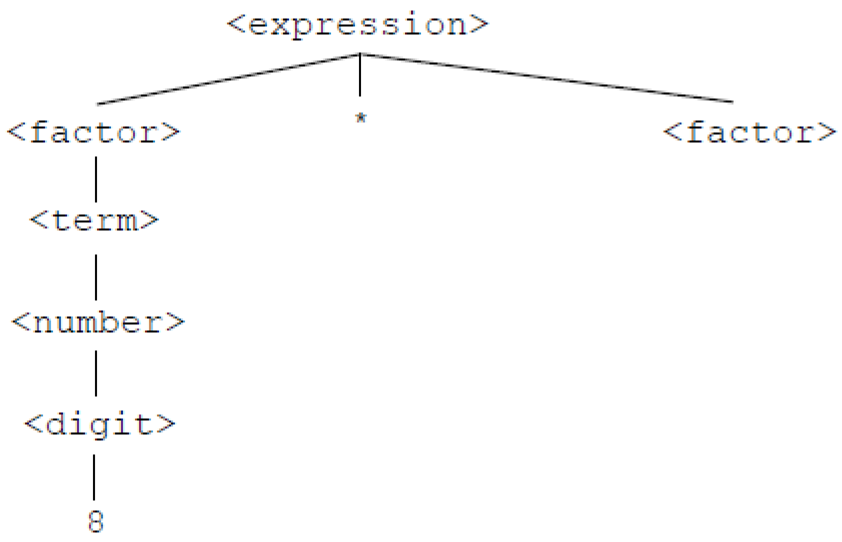
(1)

(b) Complete the table below by writing **Yes** or **No** in the empty column to indicate whether or not the strings are valid examples of the statement types from the diagram above.

Statement type	**String**	**Valid (Yes / No)**
<number>	129.376	
<factor>	23 + 17	

(2)

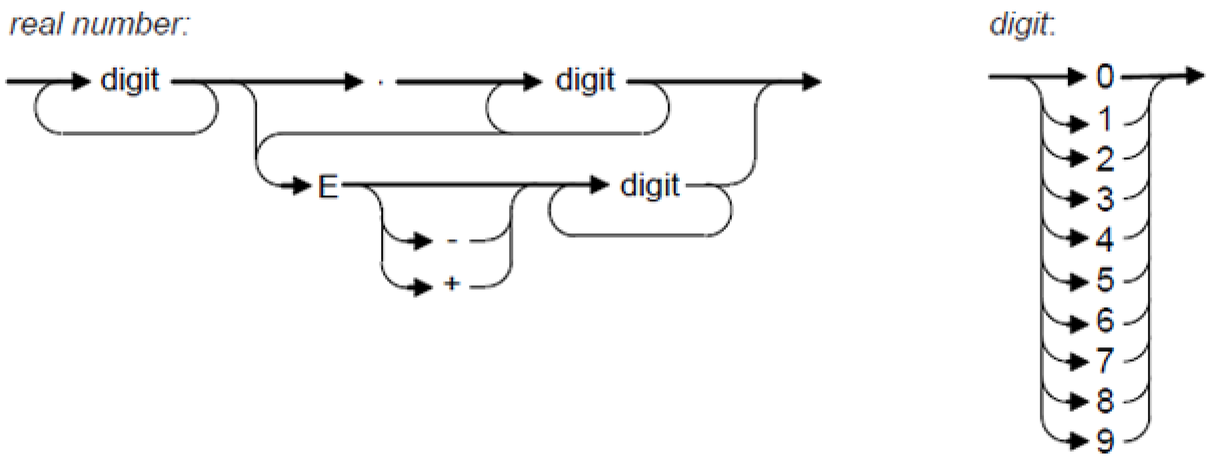
(c) A tree can be used to demonstrate that an `<expression> is valid. This is known as a parse tree.`
Complete the parse tree below to show that `8 * 4 + 21 is a valid <expression>.`



(3)
(Total 6 marks)

Q8.

In a particular programming language, the correct syntax for a real number is defined by the syntax diagrams in the diagram below.



(a) Write **Yes** or **No** in the spaces in the empty column of the table below to identify whether or not the numbers listed in the table are valid real numbers which conform to the correct syntax for this language.

Real number	Valid? (Yes / No)
203.412	
-12.87	
12.43E-12	

(3)

(b) In the same language:

A *digit* is defined as any single numeric symbol from this list: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
A *whole number* is defined as a sequence of one or more *digits*.
An *integer* is defined as a *whole number* or a + or a – symbol followed by a *whole number*.

Write Backus-Naur Form (BNF) production rules for *digit*, *whole number* and *integer*.

____

____

____

____

(3)
(Total 6 marks)