Translation Software

Question 1

6 Duraid writes a short program in a high-level programming language. An interpreter executes the program.

The following is part of Duraid's program.

```
DECLARE P, Q, R, X, Y : INTEGER
CONSTANT M = 10

P = 4
Q = 2
R = 1
X = (P + Q) * (P - Q)
Y = (M / Q) * (P + Q - R)
```

(a) Write the Reverse Polish Notation (RPN) for the following expression from Duraid's program.

```
(P + Q) * (P - Q)
```

..... [2]

(b) The interpreter is executing Duraid's program. The expressions are in infix form.

The interpreter converts the infix to RPN.

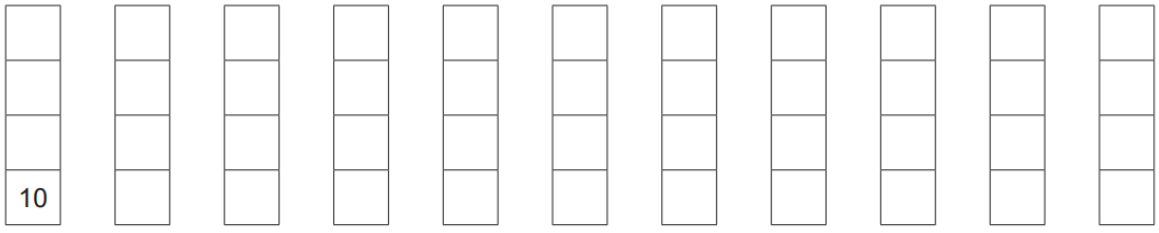
The RPN expression for `Y` is:

```
M Q / P Q + R - *
```

The interpreter evaluates this RPN expression using a stack.

(i) Show the changing contents of the stack, as the interpreter evaluates the expression for `Y`.

Use the values of the variables and constant given in the program. The first entry has been done for you.



[4]

(ii) Convert the following RPN expression back to its infix form.

```
P Q + M * R P - -
```

.....

..... [2]

(c) Explain how RPN is used by an interpreter to evaluate expressions.

.....

.....

.....

..... [2]

Question 2

4 A compiler uses a keyword table and a symbol table. Part of the keyword table is shown.

- Tokens for keywords are shown in hexadecimal.
- All of the keyword tokens are in the range 00 – 5F.

Keyword	Token
←	01
+	02
=	03
<>	04
IF	4A
THEN	4B
ENDIF	4C
ELSE	4D
REPEAT	4E
UNTIL	4F
TO	50
INPUT	51
OUTPUT	52
ENDFOR	53

Entries in the symbol table are allocated tokens. These values start from 60 (hexadecimal).

Study the following piece of pseudocode.

```
Counter ← 0
INPUT Password
REPEAT
   IF Password <> "Cambridge"
      THEN
         INPUT Password
   ENDIF
   Counter ← Counter + 1
UNTIL Password = "Cambridge"
OUTPUT Counter
```

(a) Complete the symbol table to show its contents after the lexical analysis stage.

Symbol	Token	
	Value	Type
Counter	60	Variable

[3]

(b) The output from the lexical analysis stage is stored in the following table. Each cell stores one byte of the output.

Complete the output from the lexical analysis using the keyword table **and** your answer to **part (a)**.

60	01																						

[2]

(c) The following table shows assembly language instructions for a processor which has one general purpose register, the Accumulator (ACC).

Instruction		Explanation
Op code	**Operand**	
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC
ADD	<address>	Add the contents of the given address to the ACC
STO	<address>	Store the contents of ACC at the given address

After the syntax analysis is completed successfully, the compiler generates object code.

The following lines of high level language code are compiled.

```
X = X + Y
Z = Z + X
```

The compilation produces the assembly language code as follows:

```
LDD 236
ADD 237
STO 236
LDD 238
ADD 236
STO 238
```

(i) The final stage in the compilation process that follows this code generation stage is code optimisation.

Rewrite the equivalent code after optimisation.

.....

.....

.....

.....

.....

..... [3]

(ii) Explain why code optimisation is necessary.

.....

.....

.....

..... [2]

Question 3

4 **(a)** Describe the main steps in the evaluation of a Reverse Polish Notation (RPN) expression using a stack.

.....

.....

.....

.....

.....

.....

.....

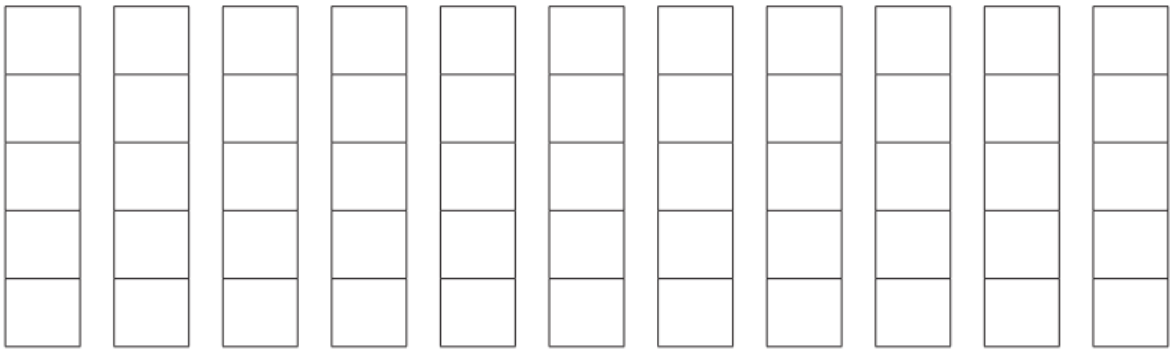
..... [4]

(b) The infix expression `8 * (5 - 2) - 30 / (2 * 3)` converts to:

`8 5 2 - * 30 2 3 * / -`

in Reverse Polish Notation (RPN).

Show the changing contents of the stack as this RPN expression is evaluated.

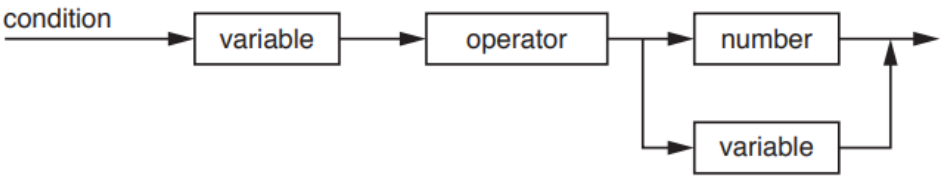


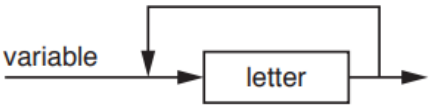
[4]

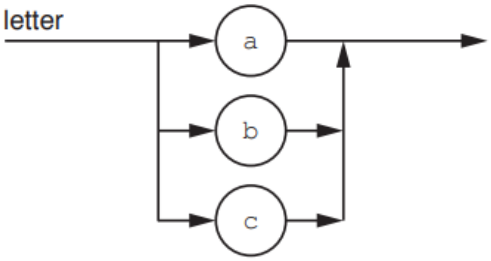
Question 4

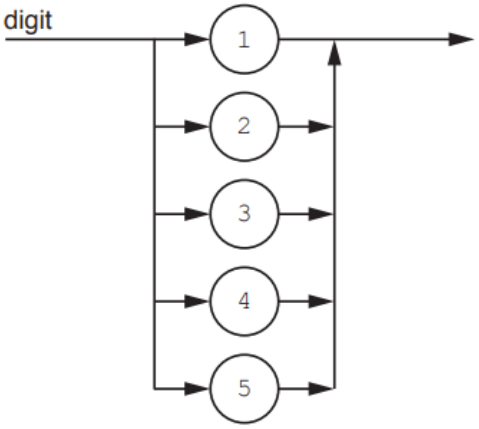
2 The following syntax diagrams for a programming language show the syntax of:

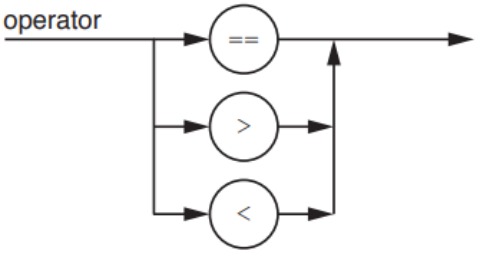
- a condition
- a variable
- a number
- a letter
- a digit
- an operator











(a) The following conditions are invalid.

Give the reason in each case.

(i) `35 > 24`

Reason .....

..... [1]

(ii) `abc := cba`

Reason .....

..... [1]

(iii) `bc < 49`

Reason .....

..... [1]

(b) Complete the Backus-Naur Form (BNF) for the syntax diagram.

<operator> ::= .....

.....

<number> ::= .....

.....

<variable> ::= .....

.....

<condition> ::= .....

.....

[6]

Question 5

7 The following are the first few lines of a source code program written in a high-level language. The source code program is to be translated by the language compiler.

```
// program written on 15 June 2019

DECLARE IsFound : Boolean;
DECLARE NoOfChildren : Integer;
DECLARE Count : Integer;
Constant TaxRate = 15;

// start of main program
For Count = 1 to 50
…
…
…
```

(a) During the lexical analysis stage, the compiler will use a keyword table and a symbol table.

(i) Identify **two** types of data in the keyword table.

Type 1 .....

Type 2 .....

[2]

(ii) Identify **two** types of data in the symbol table.

Type 1 .....

Type 2 .....

[2]

(iii) Explain how the contents of the keyword and symbol tables are used to translate the source code program.

.....

.....

.....

..... [2]

(iv) State **one** additional task completed at the lexical analysis stage that does not involve the use of a keyword or a symbol table.

.....

..... [1]

(b) The final stage of compilation can be code optimisation.

Explain why code is optimised.

.....

.....

.....

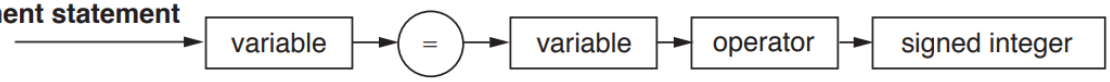
..... [2]

Question 6

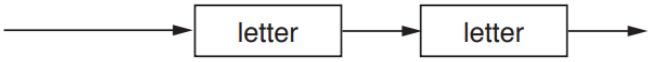
5 The following syntax diagrams show the syntax of:

- an assignment statement
- a variable
- a signed integer
- a letter
- a digit
- an operator

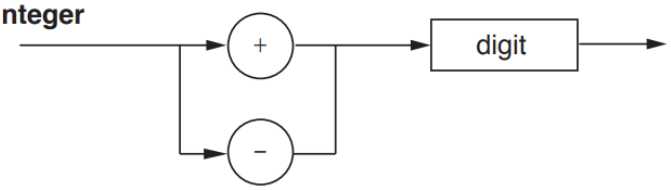
assignment statement



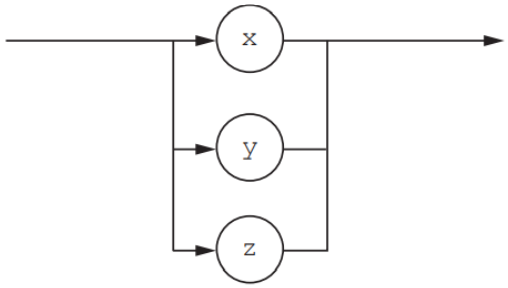
variable

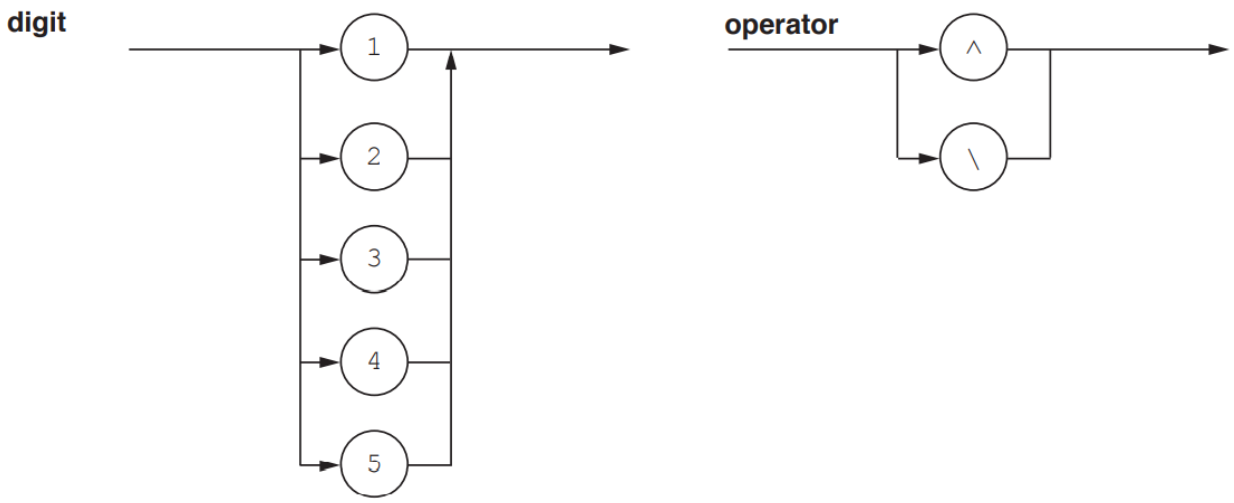


signed integer



letter





(a) The following assignment statements are invalid.

Give the reason in each case.

(i) `xy = xy ^ c4`

Reason .....

.....[1]

(ii) `zy = zy \ 10`

Reason .....

.....[1]

(iii) `yy := xz ^ - 6`

Reason .....

.....[1]

(b) Complete the Backus-Naur Form (BNF) for the syntax diagrams on the opposite page.

<assignment statement> ::=

.....

<variable> ::=

.....

<signed integer> ::=

.....

<operator> ::=

.....

[4]

(c) Rewrite the BNF rule for a variable so that it can be any number of letters.

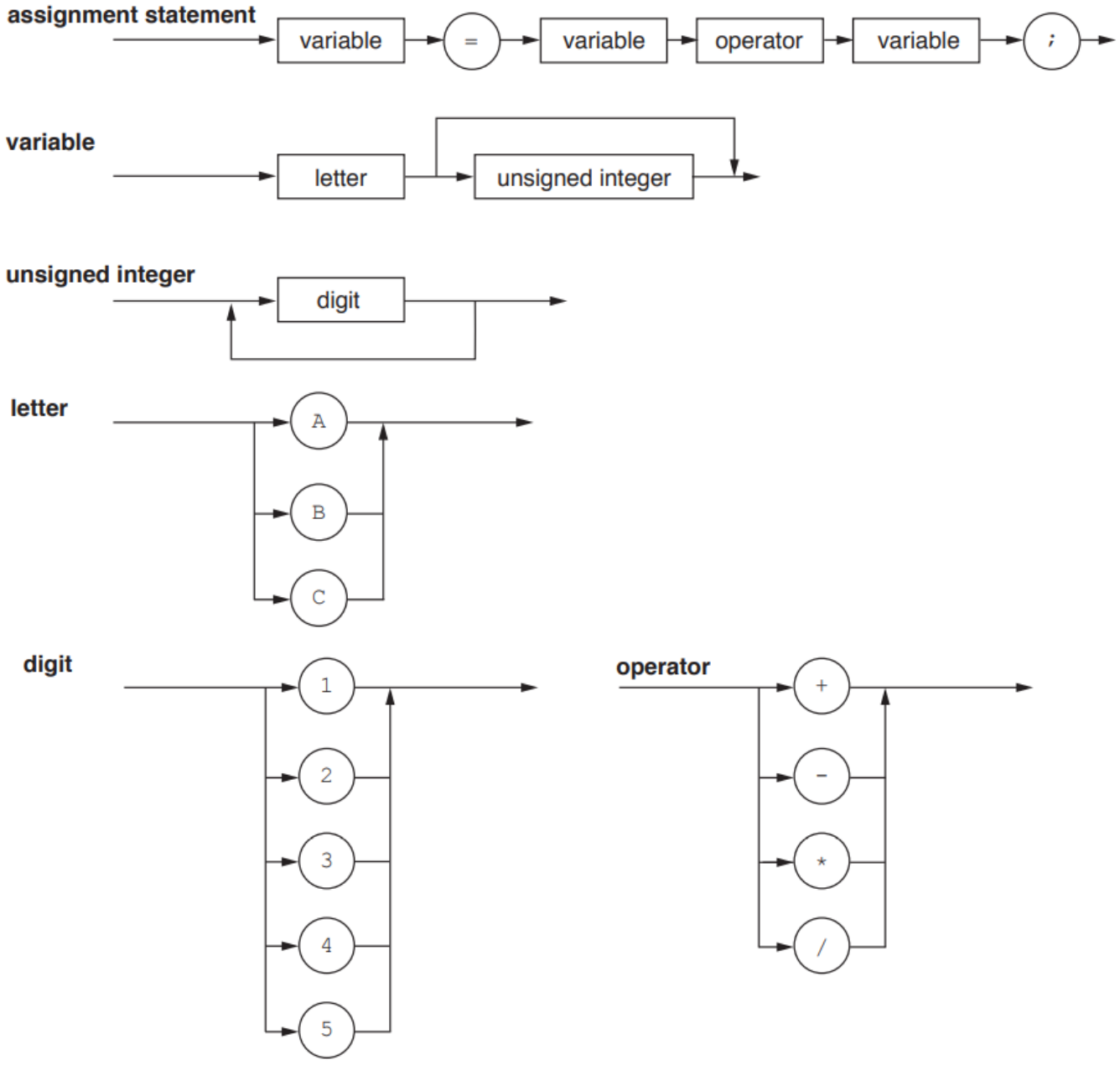
<variable> ::=

.....[2]

Question 7

2 The following syntax diagrams show the syntax of:

- an assignment statement
- a variable
- an unsigned integer
- a letter
- a digit
- an operator



(a) The following assignment statements are invalid.

Give the reason in each case.

(i) `A = B + 5;`

Reason .....

.....[1]

(ii) `A = B - D;`

Reason .....

.....[1]

(iii) `C4 = B2 - A1 + C3;`

Reason .....

.....[1]

(b) Complete the Backus-Naur Form (BNF) for the syntax diagrams shown on the opposite page.

`<assignment statement> ::=`

.....

`<variable> ::=`

.....

`<unsigned integer> ::=`

.....

`<operator> ::=`

.....

[6]

(c) The syntax of **variable** is changed to allow one or more letters followed by an unsigned integer.

Draw a syntax diagram for the new syntax of the variable.

[3]

Question 8

6 The compilation process has a number of stages. The first stage is lexical analysis.

A compiler uses a keyword table and a symbol table. Part of the keyword table is shown.

- Tokens for keywords are shown in hexadecimal.
- All of the keyword tokens are in the range `00` – `5F`.

Keyword	Token
←	01
*	02
=	03
IF	4A
THEN	4B
ENDIF	4C
ELSE	4D
FOR	4E
STEP	4F
TO	50
INPUT	51
OUTPUT	52
ENDFOR	53

Entries in the symbol table are allocated tokens. These values start from 60 (hexadecimal).

Study the following code.

```
Start ← 1
INPUT Number
// Output values in a loop
FOR Counter ← Start TO 12
    OUTPUT Number * Counter
ENDFOR
```

(a) Complete the symbol table to show its contents after the lexical analysis stage.

Symbol	Token	
	Value	Type
`Start`	`60`	Variable
`1`	`61`	Constant

[3]

(b) The output from the lexical analysis stage is stored in the following table. Each cell stores one byte of the output.

Complete the output from the lexical analysis stage. Use the keyword table and your answer to **part (a)**.

`60`	`01`														

[2]

(c) The output of the lexical analysis stage is the input to the syntax analysis stage.

Identify **two** tasks in syntax analysis.

1 .....

.....

2 .....

.....

[2]

(d) The final stage of compilation is optimisation.

(i) Code optimisation produces code that minimises the amount of memory used.

Give **one** additional reason why code optimisation is performed.

.....

.....[1]

(ii) A student uses the compiler to compile some different code.

After the syntax analysis stage is complete, the compiler generates object code.

The following lines of code are compiled.

```
X ← A + B
Y ← A + B + C
Z ← A + B + C + D
```

The compilation produces the following assembly language code.

```
LDD 236     //    loads value A to accumulator
ADD 237     //    adds value B to accumulator
STO 512     //    stores accumulator in X
LDD 236     //    loads value A to accumulator
ADD 237     //    adds value B to accumulator
ADD 238     //    adds value C to accumulator
STO 513     //    stores accumulator in Y
LDD 236     //    loads value A to accumulator
ADD 237     //    adds value B to accumulator
ADD 238     //    adds value C to accumulator
ADD 239     //    adds value D to accumulator
STO 514     //    stores accumulator in Z
```

Rewrite the assembly language code after it has been optimised.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[5]

Question 9

4 A compiler uses a keyword table and a symbol table. Part of the keyword table is shown.

- Tokens for keywords are shown in hexadecimal.
- All of the keyword tokens are in the range `00 – 5F`.

Keyword	Token
←	01
+	02
=	03
IF	4A
THEN	4B
ENDIF	4C
ELSE	4D
FOR	4E
STEP	4F
TO	50
INPUT	51
OUTPUT	52
ENDFOR	53

Entries in the symbol table are allocated tokens. These values start from `60` (hexadecimal).

Study the following code.

```
INPUT Number1
INPUT Number2
INPUT Answer
IF Answer = Number1 + Number2
   THEN
      OUTPUT 10
   ELSE
      OUTPUT 0
ENDIF
```

(a) Complete the symbol table to show its contents after the lexical analysis stage.

Symbol	Token	
	Value	Type
Number1	60	Variable
Number2	61	Variable

[3]

(b) The output from the lexical analysis stage is stored in the following table. Each cell stores one byte of the output.

Complete the output from the lexical analysis. Use the keyword table and your answer to **part (a)**.

51	60																	

[2]

(c) A student uses the compiler to compile some different code.

After the syntax analysis is complete, the compiler generates object code.

The following line of code is compiled: `X ← A + B + C - D`

The compilation produces the following assembly language code.

```
LDD 236      // loads value A into accumulator
ADD 237      // adds value B to accumulator
ADD 238      // adds value C to accumulator
STO 540      // stores accumulator in temporary location
LDD 540      // loads value from temporary location into accumulator
SUB 239      // subtracts value D from accumulator
STO 235      // stores accumulator in X
```

(i) Identify the final stage in the compilation process that follows this code generation stage.

.....[1]

(ii) Rewrite the equivalent code following the final stage.

.....

.....

.....

.....

.....

.....

.....

.....[3]

(iii) State **two** benefits of the process that is carried out in the final stage.

Benefit 1 .....

.....

Benefit 2 .....

.....

[2]

(d) An interpreter is executing a program. The program uses the variables `a`, `b`, `c` and `d`.

The program contains an expression that is written in infix form. The interpreter converts the infix expression to RPN.

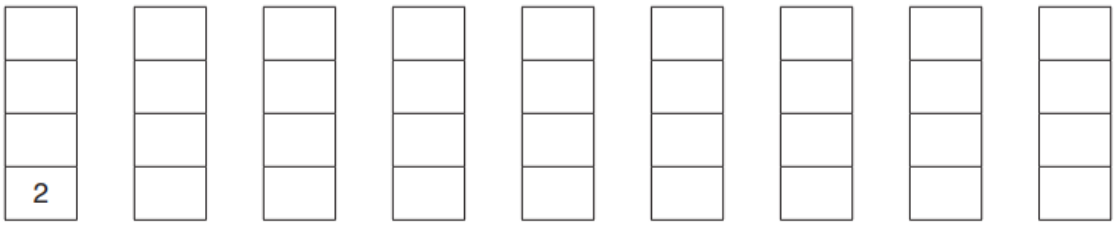
The RPN expression is: `b a c + * d + 2 -`

The interpreter evaluates this RPN expression using a stack.

The current values are: `a = 1` `b = 2` `c = 2` `d = 3`

Show the changing contents of the stack as the interpreter evaluates the expression.

The first entry on the stack has been done for you.



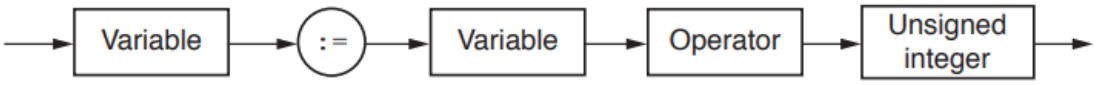
[4]

Question 10

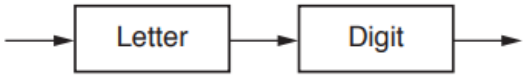
3 The following syntax diagrams for a particular programming language show the syntax of:

- an assignment statement
- a variable
- an unsigned integer
- a letter
- an operator
- a digit.

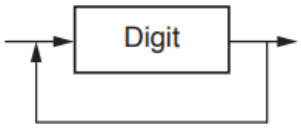
Assignment statement



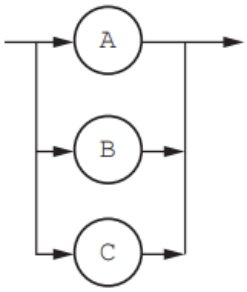
Variable



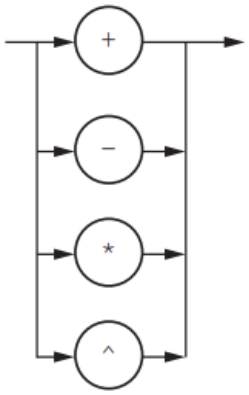
Unsigned integer



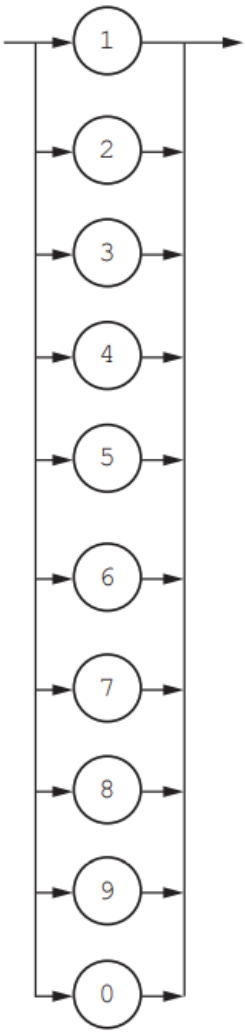
Letter



Operator



Digit



(a) The following assignment statements are invalid.

Give the reason in each case.

(i) `C2 = C3 + 123`

Reason: .....

..... [1]

(ii) `A3 := B1 - B2`

Reason: .....

..... [1]

(iii) `A32 := A2 * 7`

Reason: .....

..... [1]

(b) Complete the Backus-Naur Form (BNF) for the syntax diagrams shown.

`<digit>` has been done for you.

`<assignment_statement> ::=`

.....

`<variable> ::=`

.....

`<unsigned_integer> ::=`

.....

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

`<letter> ::=`

.....

`<operator> ::=`

.....

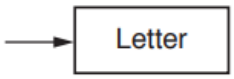
[6]

(c) The definition of <variable> is changed to allow:

- one or two letters and
- zero, one or two digits.

Draw an updated version of the syntax diagram for <variable>.

Variable



[2]

(d) The definition of <assignment_statement> is altered so that its syntax has <unsigned_integer> replaced by <real>.

A real is defined to be:

- at least one digit before a decimal point
- a decimal point
- at least one digit after a decimal point.

Give the BNF for the revised <assignment_statement> and <real>.

<assignment_statement> ::= .....

.....

<real> ::= .....

.....

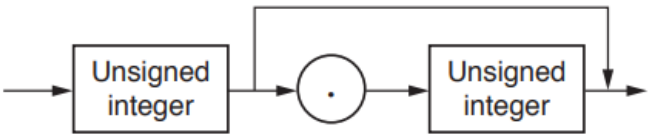
[2]

Question 11

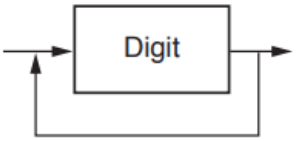
4 The following syntax diagrams for a particular programming language show the syntax of:

- an unsigned number
- an unsigned integer
- a digit.

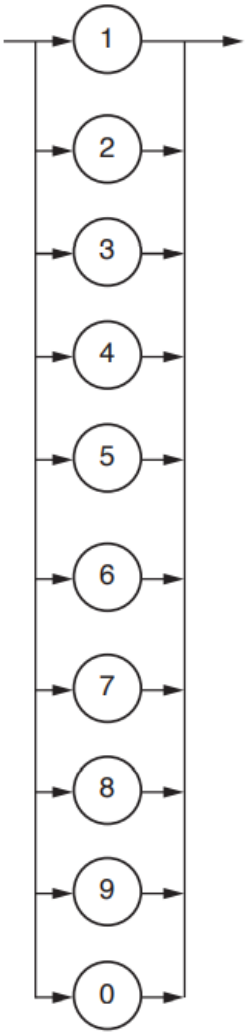
Unsigned number



Unsigned integer



Digit



(a) (i) Explain why 32 is a valid unsigned integer.

.....

.....

.....

.....[2]

(ii) Explain why 32.5 is a valid unsigned number.

.....

.....

.....

.....[2]

(b) Complete the Backus-Naur Form (BNF) for the syntax diagrams shown.

<unsigned_number> ::= .....

.....

<unsigned_integer> ::= .....

.....

<digit> ::= .....

.....
[5]

The format of an unsigned number is amended to include numbers with possible exponents.

If an unsigned number has an exponent, then the exponent part:

- will start with an 'E'
- be followed by an optional '+' or '–' sign
- and be completed by an unsigned integer.

Examples of unsigned numbers with exponents include: 3E2, 3E+3, 3E–32, 3.45E–2

(c) **(i)** Redraw the syntax diagram for unsigned number to include numbers that might have exponents.

[4]

(ii) Use your syntax diagram from **part (c)(i)** to write the BNF for an unsigned number to include numbers with exponents.

<unsigned_number> ::= .....

.....

.....

.....

.....

.....

.....

.....[4]

Question 12

2 There are four stages in the compilation of a program written in a high-level language.

(a) Four statements and four compilation stages are shown below.

Draw a line to link each statement to the correct compilation stage.

Statement	Compilation stage
This stage removes any comments in the program source code.	Lexical analysis
This stage could be ignored.	Syntax analysis
This stage checks the grammar of the program source code.	Code generation
This stage produces a tokenised version of the program source code.	Optimisation

[4]

(b) Write the Reverse Polish Notation (RPN) for the following expressions.

(i) `(A + B) * (C - D)`

..... [2]

(ii) `-A / B * 4 / (C - D)`

..... [3]

(c) An interpreter is executing a program. The program uses the variables `w`, `x`, `y` and `z`.

The program contains an expression written in infix form. The interpreter converts the infix expression to RPN. The RPN expression is:

```
x w z + y - *
```

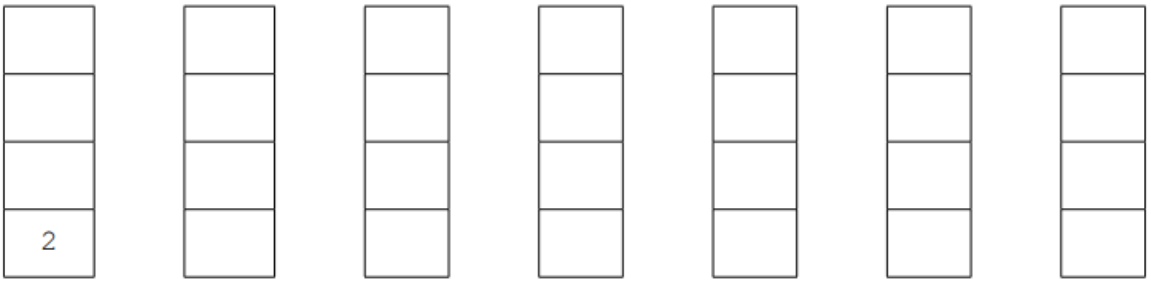
The interpreter evaluates this RPN expression using a stack.

The current values of the variables are:

```
w = 1   x = 2   y = 3   z = 4
```

(i) Show the changing contents of the stack as the interpreter evaluates the expression.

The first entry on the stack has been done for you.



[4]

(ii) Convert back to its original infix form, the RPN expression:

```
x w z + y - *
```

.....

..... [2]

(iii) Explain **one** advantage of using RPN for the evaluation of an expression.

.....

.....

.....

..... [2]

Question 13

2 There are four stages in the compilation of a program written in a high-level language.

(a) Four statements and four compilation stages are shown below.

Draw a line to link each statement to the correct compilation stage.

Statement	Compilation stage
This stage can improve the time taken to execute the statement: `x = y + 0`	Lexical analysis
This stage produces object code.	Syntax analysis
This stage makes use of tree data structures.	Code generation
This stage enters symbols in the symbol table.	Optimisation

[4]

(b) Write the Reverse Polish Notation (RPN) for the following expression.

```
P + Q - R / S
```

..... [2]

(c) An interpreter is executing a program. The program uses the variables `a`, `b`, `c` and `d`.

The program contains an expression written in infix form. The interpreter converts the infix expression to RPN. The RPN expression is:

```
b a * c d a + + -
```

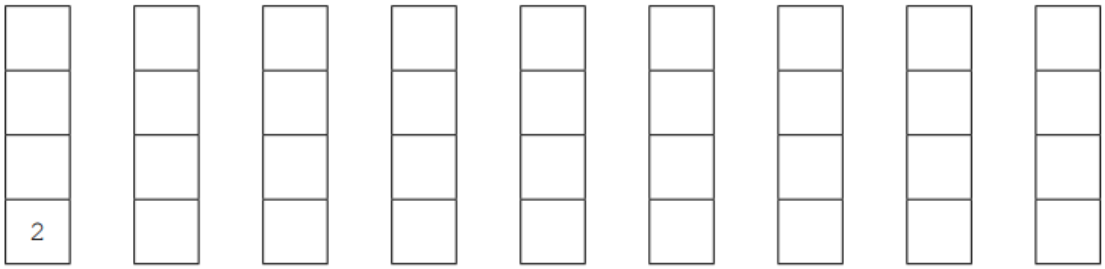
The interpreter evaluates this RPN expression using a stack.

The current values of the variables are:

```
a = 2   b = 2   c = 1   d = 3
```

(i) Show the changing contents of the stack as the interpreter evaluates the expression.

The first entry on the stack has been done for you.



[4]

(ii) Convert back to its original infix form, the RPN expression:

```
b a * c d a + + -
```

.....

..... [2]

(iii) One advantage of using RPN is that the evaluation of an expression does not require rules of precedence.

Explain this statement.

.....

.....

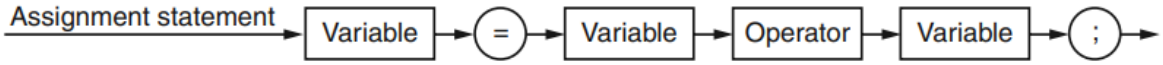
.....

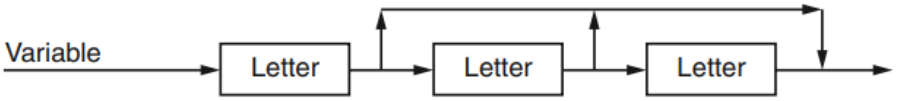
..... [2]

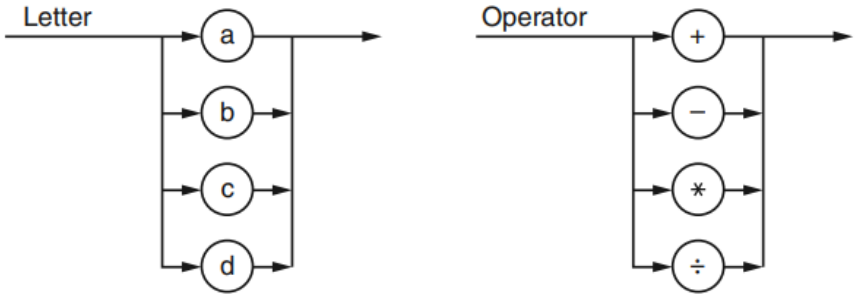
Question 14

1 The following syntax diagrams, for a particular programming language, show the syntax of:

- an assignment statement
- a variable
- a letter
- an operator







(a) The following assignment statements are invalid.

Give the reason in each case.

(i) `a = b + c`

Reason .....

.....[1]

(ii) `a = b - 2;`

Reason .....

.....[1]

(iii) `a = dd * cce;`

Reason .....

.....[1]

(b) Write the Backus-Naur Form (BNF) for the syntax diagrams shown on the opposite page.

<assignmentstatement> ::=

.....

<variable> ::=

.....

<letter> ::=

.....

<operator> ::=

.....[6]

(c) Rewrite the BNF rule for a variable so that it can be any number of letters.

<variable> ::=

.....[2]

(d) Programmers working for a software development company use both interpreters and compilers.

(i) The programmers prefer to debug their programs using an interpreter.

Give **one** possible reason why.

.....

.....[1]

(ii) The company sells compiled versions of its programs.

Give a reason why this helps to protect the security of the source code.

.....

.....[1]

Question 15

2 A compiler uses a keyword table and a symbol table. Part of the keyword table is shown below.

- Tokens for keywords are shown in hexadecimal.
- All the keyword tokens are in the range 00 – 5F.

Keyword	Token
←	01
+	02
=	03
IF	4A
THEN	4B
ENDIF	4C
ELSE	4D
FOR	4E
STEP	4F
TO	50
INPUT	51
OUTPUT	52
ENDFOR	53

Entries in the symbol table are allocated tokens. These values start from 60 (hexadecimal).

Study the following piece of code:

```
Counter ← 1.5
INPUT Num1
  // Check values
IF Counter = Num1
   THEN
      Num1 ← Num1 + 5.0
ENDIF
```

(a) Complete the symbol table below to show its contents after the lexical analysis stage.

(a) Complete the symbol table below to show its contents after the lexical analysis stage.

Symbol	Token	
	Value	Type
`Counter`	60	Variable
`1.5`	61	Constant

[3]

(b) Each cell below represents one byte of the output from the lexical analysis stage.

Using the keyword table and your answer to **part (a)** complete the output from the lexical analysis.

60	01														

[2]

(c) This line of code is to be compiled:

```
A ← B + C + D
```

After the syntax analysis stage, the compiler generates object code. The equivalent code, in assembly language, is shown below:

```
LDD 234     //loads value B
ADD 235     //adds value C
STO 567     //stores result in temporary location
LDD 567     //loads value from temporary location
ADD 236     //adds value D
STO 233     //stores result in A
```

(i) Name the final stage in the compilation process that follows this code generation stage.

.....[1]

(ii) Rewrite the equivalent code given above to show the effect of it being processed through this final stage.

.....

.....

.....

.....

.....

.....[2]

(iii) State **two** benefits of the compilation process performing this final stage.

Benefit 1 .....

.....

Benefit 2 .....

.....[2]

Question 16

2 In this question, you are shown pseudocode in place of a real high-level language. A compiler uses a keyword table and a symbol table. Part of the keyword table is shown below.

- Tokens for keywords are shown in hexadecimal.
- All the keyword tokens are in the range 00 to 5F.

Keyword	Token
←	01
+	02
=	03
IF	4A
THEN	4B
ENDIF	4C
ELSE	4D
FOR	4E
STEP	4F
TO	50
INPUT	51
OUTPUT	52
ENDFOR	53

Entries in the symbol table are allocated tokens. These values start from 60 (hexadecimal).

Study the following piece of code:

```
Start ← 0.1
// Output values in loop
FOR Counter ← Start TO 10
    OUTPUT Counter + Start
ENDFOR
```

(a) Complete the symbol table below to show its contents after the lexical analysis stage.

Symbol	Token	
	Value	Type
`Start`	60	Variable
`0.1`	61	Constant

[3]

(b) Each cell below represents one byte of the output from the lexical analysis stage.

Using the keyword table and your answer to **part (a)** complete the output from the lexical analysis.

60	01												

[2]

(c) The compilation process has a number of stages. The output of the lexical analysis stage forms the input to the next stage.

(i) Name this stage.

.....[1]

(ii) State **two** tasks that occur at this stage.

.....

.....

.....

.....[2]

(d) The final stage of compilation is optimisation. There are a number of reasons for performing optimisation. One reason is to produce code that minimises the amount of memory used.

(i) State another reason for the optimisation of code.

.....[1]

(ii) What could a compiler do to optimise the following expression?

```
A ← B + 2 * 6
```

.....

.....

.....[1]

(iii) These lines of code are to be compiled:

```
X ← A + B
Y ← A + B + C
```

Following the syntax analysis stage, object code is generated. The equivalent code, in assembly language, is shown below:

```
LDD 436     //loads value A
ADD 437     //adds value B
STO 612     //stores result in X
LDD 436     //loads value A
ADD 437     //adds value B
ADD 438     //adds value C
STO 613     //stores result in Y
```

(iv) Rewrite the equivalent code, given above, following optimisation.

.....

.....

.....

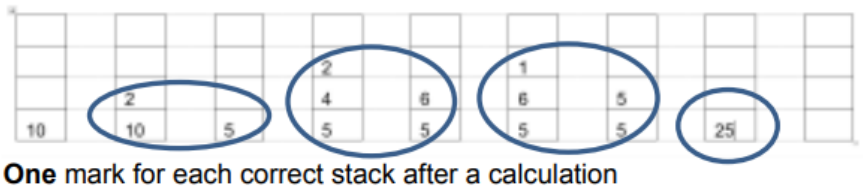
.....

.....

.....[3]

Answer

Answer 1

6(a)	`P Q + P Q - *` **One** mark for `P Q +` **One** mark for `P Q - *`	**2**
6(b)(i)	 **One** mark for each correct stack after a calculation	**4**
6(b)(ii)	`((P + Q) * M) - (R - P)` **One** mark for `((P + Q) * M)` **One** mark for `- (R - P)`	**2**
6(c)	Any **two** from: • Expressions are always evaluated left to right • Each operator uses the two previous values on the stack (except unary minus) • Description of pushing and popping on a stack	**2**

Answer 2

4(a)	**1 mark** for **2** correct rows, **2 marks** for **3** correct rows, **3 marks** for **4** correct rows	**3**

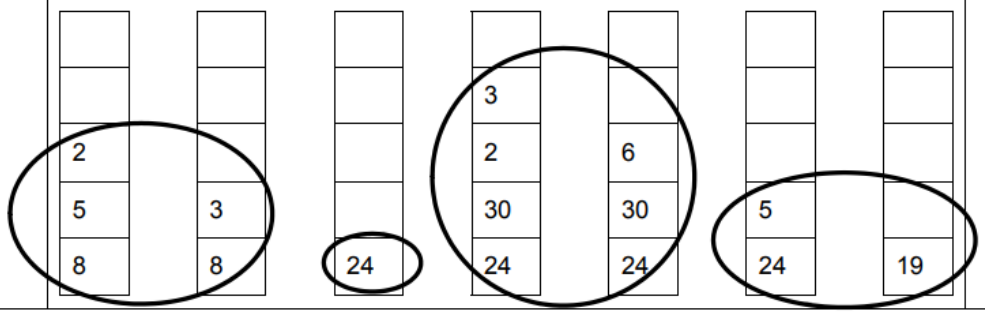
Symbol	Token	
	Value	Type
Counter	60	Variable
0	61	Constant
Password	62	Variable
"Cambridge"	63	Constant
1	64	Constant

4(b)	<table><tr><td>60</td><td>01</td></tr></table> First two cells given in question. **1 mark** for next 3 cells <table><tr><td>61</td><td>51</td><td>62</td></tr></table> **1 mark** for the remainder <table><tr><td>4E</td><td>4A</td><td>62</td><td>04</td><td>63</td><td>4B</td><td>51</td><td>62</td><td>4C</td><td>60</td></tr></table> .... <table><tr><td>01</td><td>60</td><td>02</td><td>64</td><td>4F</td><td>62</td><td>03</td><td>63</td><td>52</td><td>60</td></tr></table>	**2**

4(c)(i)	**1 mark** per bullet point
	∞ Removing the fourth line (LDD 238) …
	∞ Changing operand for second ADD from 236 to 238 …
	∞ … First three lines and last line unchanged
	
	`LDD 236`
	`ADD 237`
	`STO 236`
	`ADD 238`
	`STO 238`
		3

4(c)((ii)	**1 mark** per bullet point (**max 2**) ∞ Optimisation means that the code will have fewer instructions ∞ Optimised code occupies less space in memory ∞ Fewer instructions reduces the execution time of the program	**2**

Answer 3

4(a)	**1 mark** per bullet point (max 4) ∞ Working from left to right in the expression ∞ If element is a number PUSH that number onto the stack ∞ If element is an operator then POP the first two numbers from stack … ∞ … perform that operation on those numbers ∞ PUSH result back onto stack ∞ End once the last item in the expression has been dealt with	**4**
4(b)	**1 mark** per ring (not all stacks are shown) Do not allow operators in stacks Accept intermediate correct stack values 	**4**

Answer 4

2(a)(i)	`35` is not a variable	**1**
2(a)(ii)	`:=` is not an operator	**1**
2(a)(iii)	`9` is not a digit	**1**

2(b)	**1 mark** for each bullet point
	
	`<operator>::=`
	• `==` | `>` | `<`
	
	`<number>::=`
	• `<digit><digit>`
	
	`<variable>::=`
	• `<letter>`
	• |`<letter><variable>`
	
	`<condition>::=`
	• `<variable><operator><number>`
	• |`<variable><operator><variable>`
		6

Answer 5

7(a)(i)	1 mark for each bullet point to **max 2** Keyword table: • The reserved words used • The operators used • Their matching tokens	**2**
7(a)(ii)	1 mark for each bullet point to **max 2** Symbol table: • Identifier name used • … the (data) type • … role, e.g. variable, constant, array, procedure / scope • Location (marker) // value of constant	**2**
7(a)(iii)	1 mark per bullet point to **max 2** • Keywords / operators are looked up (in the keyword table) • Keywords / operators are represented by tokens • Identifiers are looked up in (the symbol table) • Identifiers are converted to locations / addresses • Used to create a sequence of tokens (for the program)	**2**

7(a)(iv)	The white space removed // redundant characters are removed // removal of comments // identification of errors	**1**
7(b)	1 mark per bullet point to **max 2** • Redundant code removed // fewer instructions required • Program requires less memory / storage space • Code reorganised to make it more efficient • Program will complete task in a shorter time	**2**

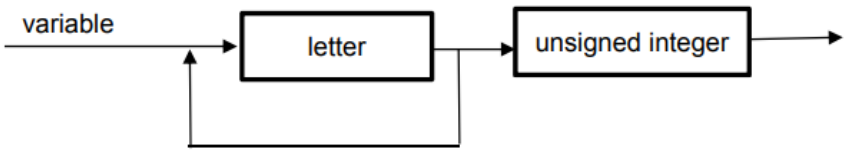
Answer 6

5(a)(i)	`c4` is not a signed integer	**1**
5(a)(ii)	`10` is not a valid signed integer // 0 is not a valid digit/signed integer // only one digit allowed	**1**
5(a)(iii)	wrong assignment operator // should be = not := // 6 is not a valid digit/signed integer	**1**
5(b)	1 mark per bullet assignment ∞ <variable>=<variable><operator><signed integer> variable ∞ <letter><letter> signed integer ∞ +<digit> | -<digit> operator ∞ ^ | \ <assignment statement> ::= <variable> = <variable><operator><signed integer> <variable> ::= <letter><letter> <signed integer> ::= +<digit> | -<digit> <operator> ::= ^ | \	**4**
5(c)	1 mark per bullet ∞ <letter>| ∞ <letter><variable> For example: <letter>|<letter><variable> <letter>|<variable><letter>	**2**

Answer 7

2(a)(i)	`5` is not a variable	**1**
2(a)(ii)	`D` is not a valid letter	**1**
2(a)(iii)	There are two operators (only one is allowed) // three variables on the right hand side but only two allowed	**1**

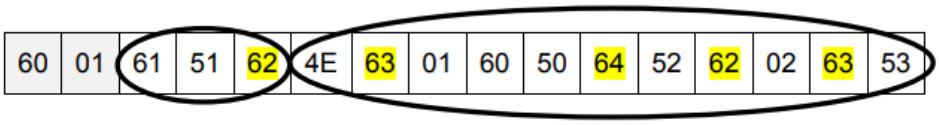
2(b)	**1** mark for each bullet
	
	assignment:
	• `<variable> = <variable><operator><variable>;`
	
	variable:
	• `<letter>|`
	• `<letter><unsigned integer>`
	
	unsigned integer:
	• `<digit>|`
	• `<digit><unsigned integer>`
	
	operator:
	• `+ | - | * | /`
	
	`<assignment statement> ::= <variable> = <variable><operator><variable>;`
	`<variable> ::= <letter> | <letter><unsigned integer>`
	`<unsigned integer> ::= <digit> | <digit><unsigned integer>`
	`<operator> ::= + | - | * | /`
		6

2(c)	**1** mark per bullet • variable with arrow • followed by repeated letter • followed by unsigned integer and arrow 	**3**

Answer 8

<table>
<tr><td>6(a)</td><td>1 mark for each correct row

<table><tr><th rowspan="2">Symbol</th><th colspan="2">Token</th></tr><tr><th>Value</th><th>Type</th></tr><tr><td>Start</td><td>60</td><td>Variable</td></tr><tr><td>1</td><td>61</td><td>Constant</td></tr><tr><td>Number</td><td>62</td><td>Variable</td></tr><tr><td>Counter</td><td>63</td><td>Variable</td></tr><tr><td>12</td><td>64</td><td>Constant</td></tr></table></td><td>3</td></tr>
<tr><td>6(b)</td><td>1 mark for each circled section

  </td><td>2</td></tr>
<tr><td>6(c)</td><td>1 mark per bullet point to max 2:

∞ constructing parse tree // parsing
∞ checking the table of tokens to ensure that the rules/syntax/grammar of the language are/is obeyed
∞ producing an error report</td><td>2</td></tr>
<tr><td>6(d)(i)</td><td>shortens execution time of program// time taken to execute whole program decreases</td><td>1</td></tr>
<tr><td>6(d)(ii)</td><td>1 mark for each of the following:

∞
<pre>LDD 236
ADD 237
STO 512
ADD 238
STO 513
ADD 239
STO 514</pre>
∞ Remove line 4 <code>LDD 236</code> correct lines 3 and 6 in original code
∞ Remove line 5 <code>ADD 237</code> correct lines 3 and 6 in original code
∞ Remove line 8 and 9 <code>LDD 236</code> and <code>ADD 237</code> correct lines 7 and 11 in original code
∞ Remove line 10 <code>ADD 238</code> correct lines 7 and 11 in original code</td><td>5</td></tr>
</table>

Answer 9

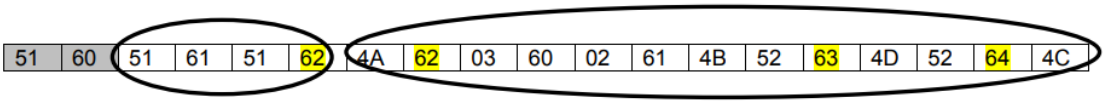
4(a)

1 mark per row

Symbol	Token	
	Value	Type
`Number1`	60	Variable
`Number2`	61	Variable
`Answer`	62	Variable
`10`	63	Constant//Literal
`0`	64	Constant//Literal

4(b)

1 mark for each circled section



4(c)(i)

(Code) Optimisation

4(c)(ii)

1 mark per bullet point:

-
```
LDD 236
ADD 237
ADD 238
SUB 239
STO 235
```
Copy the instructions

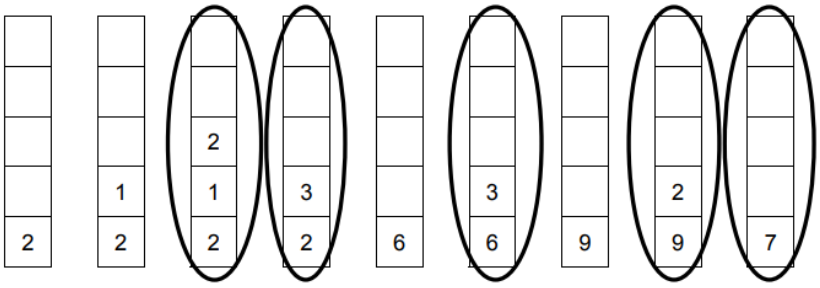
- Remove line 4 `STO 540` correct lines 3 and 6 in original code
- Remove line 5 `LDD 540` correct lines 3 and 6 in original code

4(c)(iii)

1 mark per bullet point:

- Code has fewer instructions/occupies less space in memory
- shortens execution time of program // time taken to execute whole program decreases

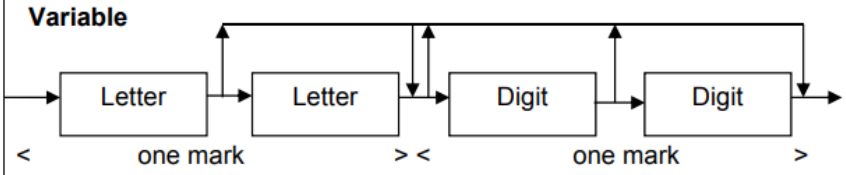
4(d)



1 mark ← 1 mark → 1 mark

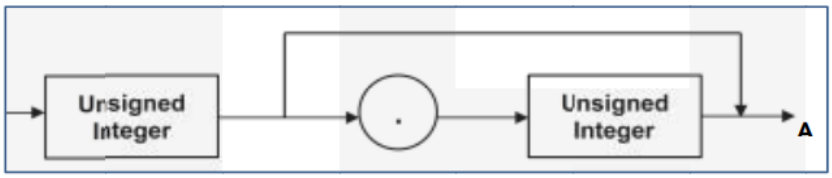
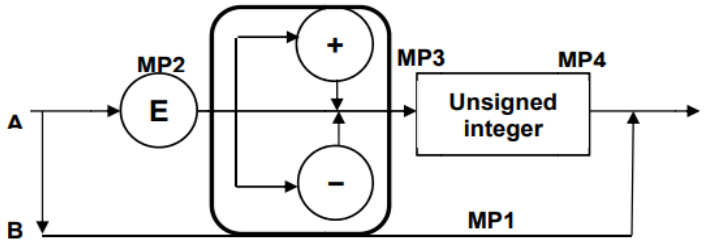
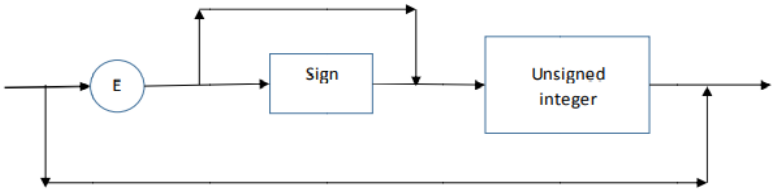
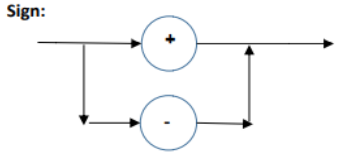
1 mark no operators on the stack anywhere

Answer 10

3(a)(i)	There should be a colon before the '=' sign	**1**
3(a)(ii)	The second operand should be an unsigned integer and not a variable	**1**
3(a)(iii)	A32 is not a variable, as a variable should be a letter followed by a single digit	**1**
3(b)	`<assignment_statement> ::= <variable> :=` 1 `<variable> <operator> <unsigned_integer>` 1 `<variable> ::= <letter> <digit>` 1 `<unsigned_integer> ::= <digit> |` 1 `<digit> <unsigned_integer>` 1 `<letter> ::= A | B | C` `<operator> ::= + | - | * | ^` 1	**6**
3(c)	**Variable**  Syntax diagram shows one or two letters 1 Syntax diagram shows zero, one or two digits 1	**2**
3(d)	`<assignment_statement> ::=` `<variable> := <variable> <operator> <real>` 1 `<real> ::= <unsigned_integer> . <unsigned_integer>` 1	**2**

Answer 11

4(a)(i)	Because a valid unsigned integer can be two digits / one or more digits (1) Both 3 and 2 are digits (1)	**2**
4(a)(ii)	Because a valid unsigned number can be an unsigned integer followed by a decimal point followed by an unsigned integer (1) 32 is an unsigned integer and 5 is an unsigned integer (because it is a digit) and there is a point in between (1) Alternative response for 2 marks, combination of order and validity: 32 is a (valid) unsigned integer, followed by a decimal point, and 5 which is another (valid) unsigned integer Validity mark must refer to 32 and 5	**2**

4(b)	<unsigned number> ::= **<unsigned_integer> |** (1) **<unsigned_integer>.<unsigned_integer>** (1) Accept order reversed: <unsigned_integer> ::= **<digit> |** (1) **<digit> <unsigned_integer>** (1) Accept **<digit> |<unsigned_integer> <digit>** If order reversed mark as above <digit> ::= **1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0** (1) Accept the list in any order, as long as all 10 digits included	**5**
4(c)(i)	  **MP1**: Line to indicate exponent is optional (B line) (1) **MP2**: 'E' present at start of exponent (1) **MP3**: Optional '+' or '-' (1) **MP4**: Unsigned integer (1) Alternative correct answer: MP3 needs both the sign 'box' and the sign diagram for the mark  	**4**

4(c)(ii)	<unsigned number> ::= **<unsigned_integer >** **| <unsigned integer>.<unsigned_integer>** (1) Accept any order **| <unsigned_integer > <exponent>** **| <unsigned integer>.<unsigned_integer> <exponent>** (1) Accept any order **<exponent> ::= E <sign> <unsigned_integer>** **| E <unsigned integer>** (1) **<sign> ::= + | -** (1)	**4**

Answer 12

2 (a)

Statement	Compilation stage
This stage removes any comments in the program code	Lexical analysis
This stage could be ignored	Syntax analysis
This stage checks the grammar of the program code	Code generation
This stage produces a tokenised version of the program code	Optimisation

1 mark for each correct line

[4]

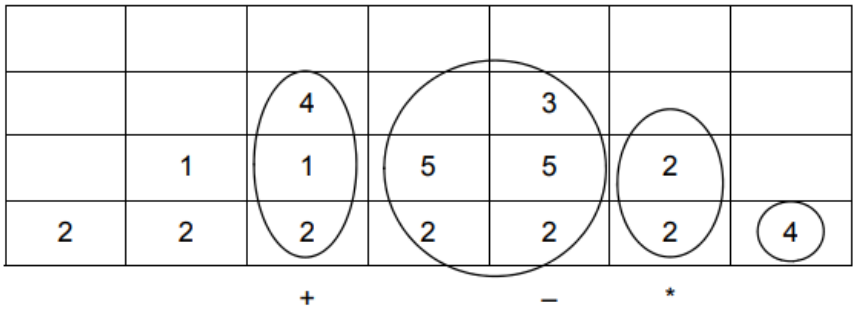
(b) (i) A B +
C D – *

[1]
[1]

(ii) A –
B / 4 *
C D – /

[1]
[1]
[1]

(c) (i)



1 mark per ring

[4]

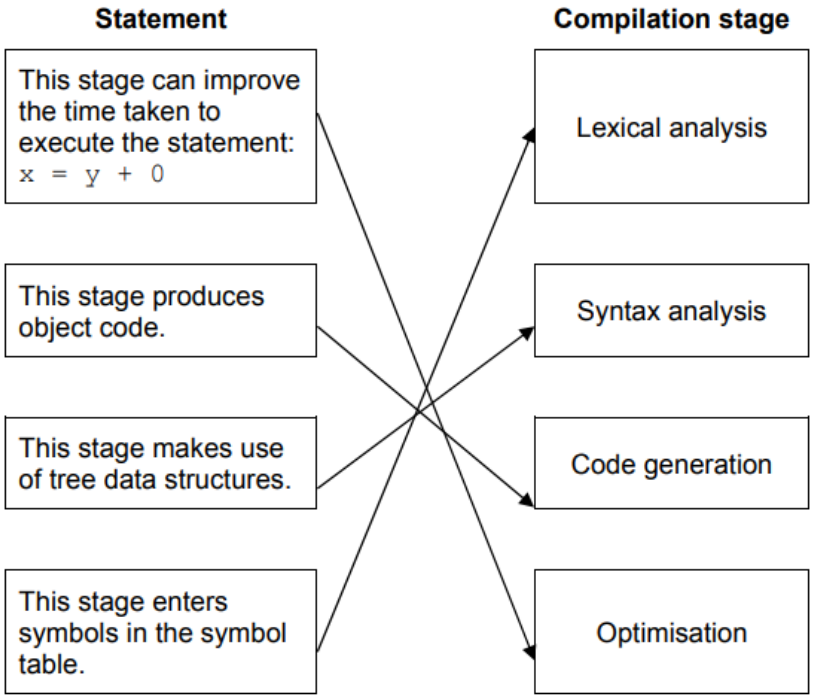
(ii) x * [1]
(w + z – y) [1]
Order must be correct for both parts

(iii) No need for rules of precedence [1]
No need for brackets [1]
In RPN evaluation of operators is always left to right [1]

[Max 2]

Answer 13

2 (a)

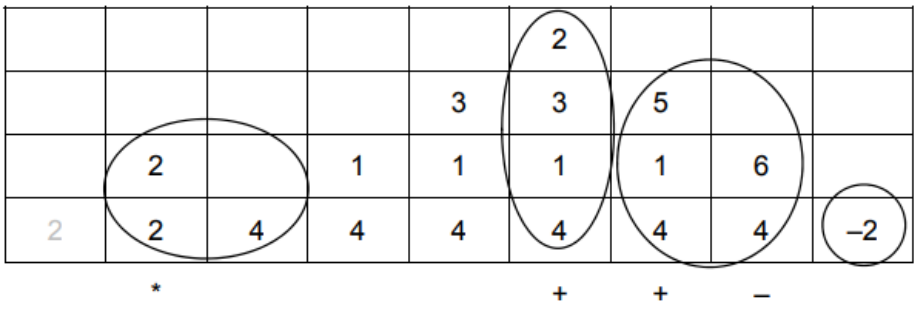


1 mark for each correct line

[4]

(b) P Q + [1]
R S / – [1]

(c) **(i)**



1 mark per ring

[4]

(ii) b * a [1]
– (c + d + a) [1]
Order must be correct for both parts

(iii) Rules of precedence means different operators have different priorities // by example multiply is done before add [1]
In RPN evaluation of operators is left to right // operators are used in the sequence in which they are read [1]
No need for brackets // infix may require the use of brackets [1]

[Max 2]

Answer 14

1 (a) (i)	';' missing	1
(ii)	'2' is not a variable	1
(iii)	'e' is not a valid letter	1
(b)	<assignment statement> ::= <variable> = <variable><operator><variable>; <variable> ::= <letter>|<letter><letter> |<letter><letter><letter> <letter> ::= a|b|c|d <operator> :: =+|-|*|÷	2 2 1 1

(c)	<letter> | <letter><variable> // <letter> | <variable><letter>	2

(d) (i)	debugging is faster / easier
	// can debug incomplete code
	// better diagnostics
		1

(ii)	compiler produces executable version – not readable / no need for source code // difficult to reverse-engineer	1

Answer 15

2 (a)

Symbol	Token	
	Value	Type
Counter	60	variable
1.5	61	constant
Num1	62	variable
5.0	63	constant

[1]

[1+1]

(b)

60	01	61	51	62	4A	60	03	62	4B	62	01	62	02	63	4C

[1+1]

(c) (i) Code optimisation [1]

(ii) LDD 234
ADD 235 [1]
ADD 236
STO 233 [1]

1 mark for first 2 lines, 1 mark for last 2 lines, with no other lines added

(iii) Code has fewer instructions / occupies less space in memory when executed [1]
minimises execution time of code // code will execute faster [1]

Answer 16

2 (a)

Symbol	Token	
	Value	Type
Start	60	Variable
0.1	61	Constant
Counter	62	Variable
10	63	Constant

[1]

[1+1]

(b)

60	01	61	4E	62	01	60	50	63	52	62	02	60	53

[1+1]

(c) (i) syntax analysis [1]

(ii) any **two** points from:

construct parse tree // parsing
checking syntax/grammar
produce error report [max. 2]

(d) (i) Minimise the execution time // code runs faster [1]

(ii) Compiler could calculate 2*6 and replace it with the value 12. [1]

(iii)

```
LDD 436
ADD 437
STO 612
ADD 438
STO 613
```

} [1]

[1]
[1]

–1 for each additional instruction; 0 for copy of original code