

Translation Software

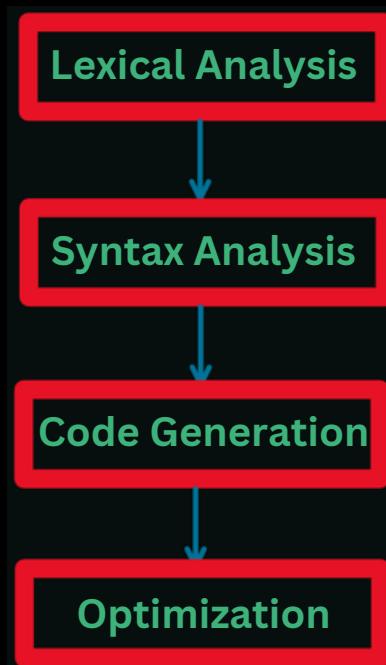


Papers Dock

COMPUTER SCIENCE 9618 PAPER 3

Translation Software

Stages In The Compilation Of A Program



Stage 1 : Lexical Analysis

- Unnecessary Characters (White Space, Comments are removed)
- Source Program is converted to tokens

Tokenization

```
Counter = 0  
For Count = 1 to 50  
    Print "Hello"  
End For
```

Tokens : Any variable, consonant, keyword, symbol, FOR, Count, Print, =, <--

Total Number Of Tokens Are : 12

Keyword Table

- Reserved word (FOR TO PRINT)
- Operators
- Their matching tokens

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

Symbol Table

- Identifiers (Variable, Consonant) name used
- The data type
- role (Variable, Consonant, array, procedure)
- Value / Location

Symbol	Token	
	Value	Type
Start	60	Variable
1	61	Constant
Number	62	Variable
Counter	63	Variable
12	64	Constant

A Keyword Table is a predefined list of reserved words in a programming language, such as if, while, and int, used by the compiler to recognize language constructs. The Symbol Table, on the other hand, stores information about user-defined symbols like variables and functions, including their data types, scope, and memory locations.

Exam Style Question

- 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]

Question : How the contents of the keyword and symbol tables are used to translate the source code program ?

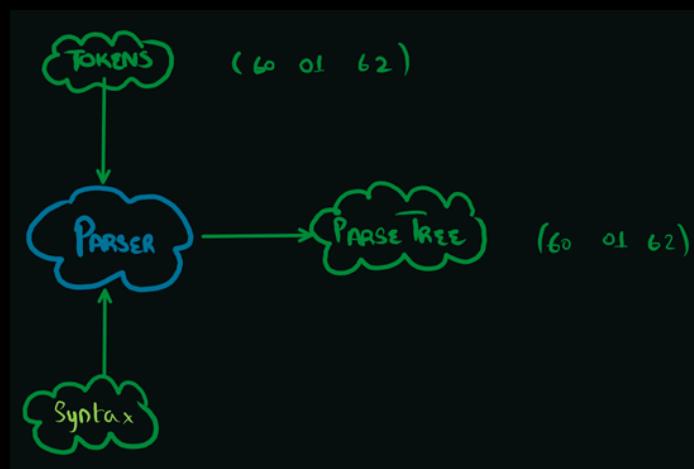
- **Keywords/operators are looked up (in the keyword table)**
- **Keywords/operators are represented by the tokens**
- **Identifiers are looked up in (the symbol table)**
- **Identifiers are converted to locations/addresses**
- **Used to create a sequence of tokens (for the program)**

Question : Additional tasks completed at the lexical analysis stage that does not involve the use of a keyword or a symbol table:

- **White space removed**
- **Redundant characters are removed**
- **Removal of comments**

Stage 2 : Syntax Analysis

- Constructing Parse Tree
- Checking the table of tokens to ensure that the rules / syntax / grammar of the language are obeyed
- Producing an error report



The process of working out if a given statement is valid according to the given BNF production rules is called parsing

The grammatical rules or syntax for a programming language need to be set out clearly so a programmer can write code that obeys the rules, and a compiler can be built to check that a program obeys these rules.

The rules can be shown graphically in a SYNTAX DIAGRAM or using a meta language such as Backus-Naur Form (BNF) notation.

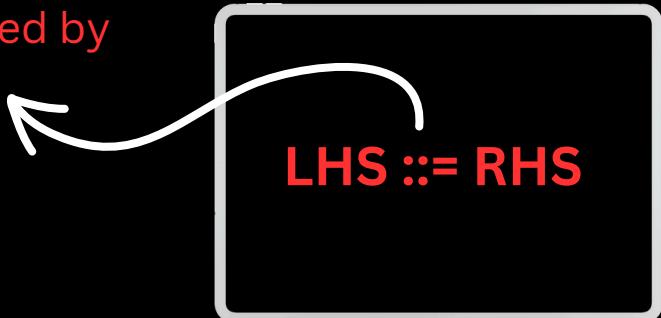
Question : Outline the purpose of syntax analysis?

- **It checks that the code matches the grammar of the language**
- **Syntax errors are reported**
- **A parse tree is produced.**

Backus Naur Form

Is a mathematical way of defining the syntax

is defined by



note : |
represents OR

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

<Number> ::= <DIGIT><DIGIT>

<Number> ::= <DIGIT> | <DIGIT><Number>

<Letter> ::= A | B | C | D | E | F | G | H | I | J | K ...

A word consists of two letters

<word> ::= <Letter><Letter>

A word consists of any number of letters

<word> ::= <Letter> | <Letter><word>

A digit could be 1 2 3 4 5

<Digit> ::= 0 | 1 | 2 | 3 | 4 | 5

Operator could be + - / *

<operator> ::= + | - | / | *

Letter could be A B C D E F

<Letter> ::= A | B | C | D | E | F

Variable could be a single letter or letter followed by a single digit.

<Variable> ::= <Letter> | <Letter><Digit>

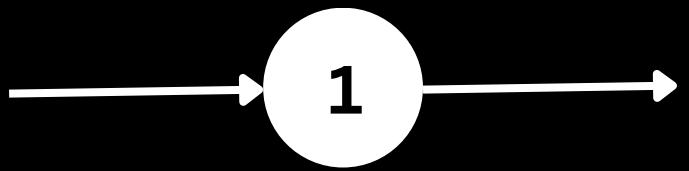
Assignment Statement

Number = N1 + N2;

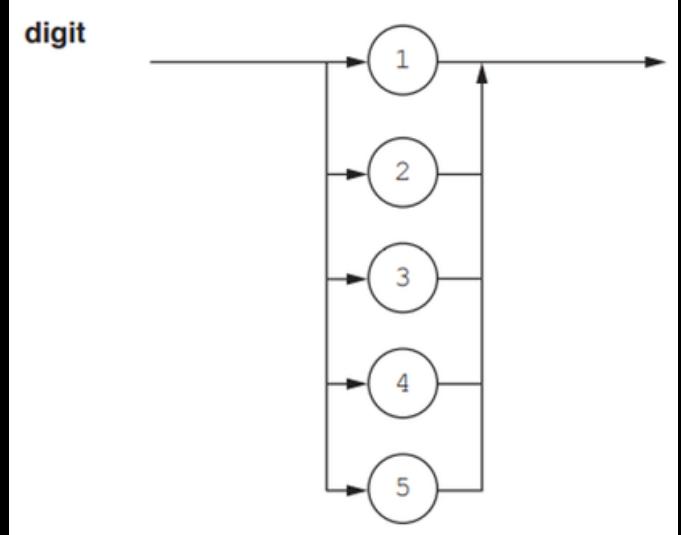
<AssignmentStatement> ::= <Variable> = <Variable><operator><Variable> ;

Syntax Diagram

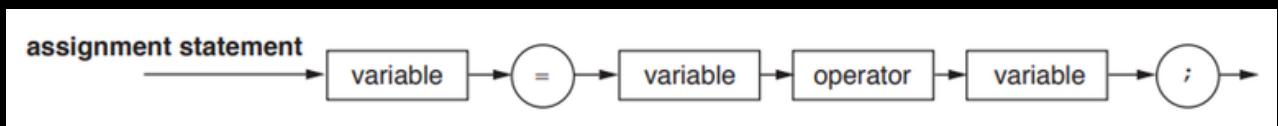
<DIGIT> ::= 1



<DIGIT> ::= 1 | 2 | 3 | 4 | 5



AssignmentStatement ::= <variable> = <variable><operator> <variable>;



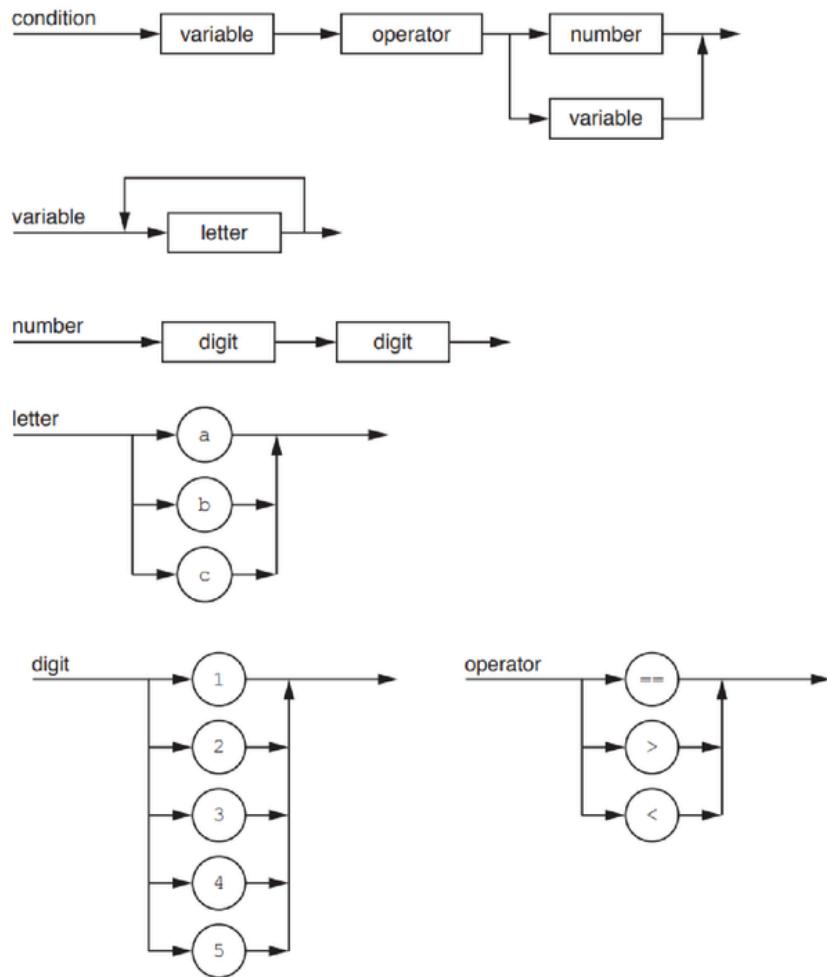
<word> ::= <Letter> | <Letter> <word>

<word> ::= <Letter> | <Letter> <digit>

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.

$\langle \text{operator} \rangle ::= == | > | <$
.....

$\langle \text{number} \rangle ::= <\text{Digit}> <\text{Digit}>$
.....

$\langle \text{variable} \rangle ::= <\text{Letter}> | <\text{Letter}> <\text{Variable}>$
.....

$\langle \text{condition} \rangle ::= <\text{Variable}> <\text{Operator}> <\text{number}> |$
 $<\text{Variable}> <\text{Operator}> <\text{Variable}>$
.....

[6]

Stage 3 : Code Generation

- The code generation stage produces an object code.
- The program should have correct syntax.
- The object code is in machine-readable form.

Stage 4 : Code Optimization

Why code Optimization is necessary ?

- Optimization means that the code will have fewer instructions.
- Optimized code occupies less space in memory.
- Fewer instructions reduce the execution time of the program

Original code	$w = x + y$ $v = x + y + z$	Object code	LDD x ADD y STO w LDD x ADD y ADD z STO v
Optimised code	$w = x + y$ $v = w + z$	Object code	LDD x ADD y STO w ADD z STO v

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

A \leftarrow 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.

Code Optimization [1]

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

LDD 234

ADD 235

ADD 236

STO 233

.....

..... [2]

Benefits of optimizing the code

- Code has fewer instructions/occupies less space in memory.
- Time taken to execute the whole program decreases.

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

Explain why code is optimised.

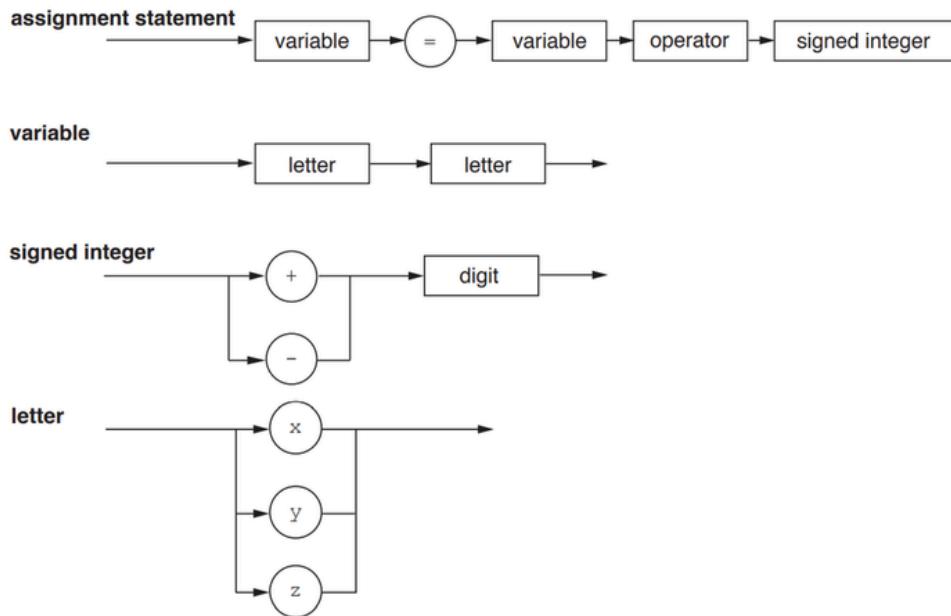
.....
.....
.....
.....

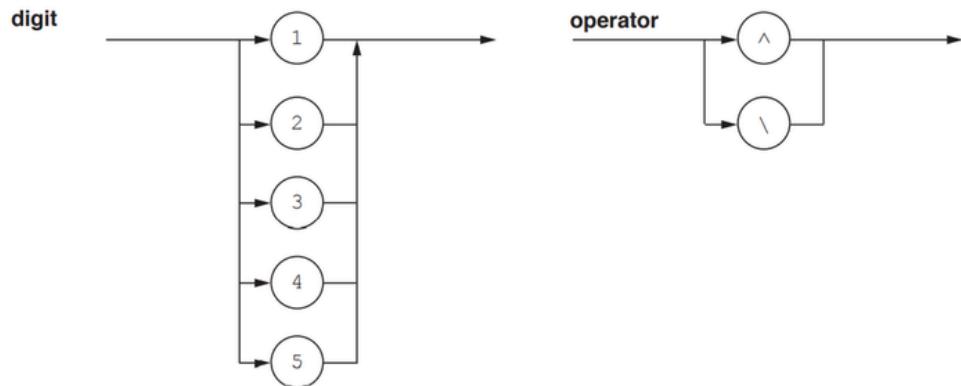
[2]

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





- (a) The following assignment statements are invalid.

Give the reason in each case.

(i) $xy = xy \wedge c4$

Reason [1]

(ii) $zy = zy \backslash 10$

Reason [1]

(iii) $yy := xz \wedge - 6$

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]

Reverse Polish Notation

Is a method of representing an arithmetic expression

Infix Form : A - B

Reverse Polish Notation : AB-

Case 1 : Using RPN to evaluate the expression using stacks

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

The interpreter converts the infix to RPN.

$$M = 10$$

The RPN expression for Y is:

$$P = 4$$

M Q / P Q + R - *

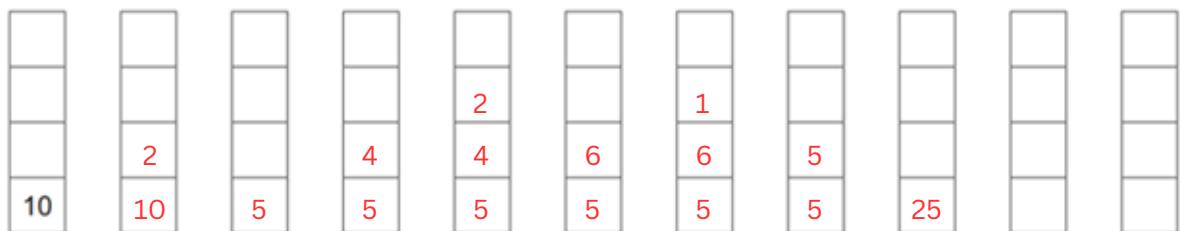
$$Q = 2$$

The interpreter evaluates this RPN expression using a stack.

$$R = 1$$

- (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.



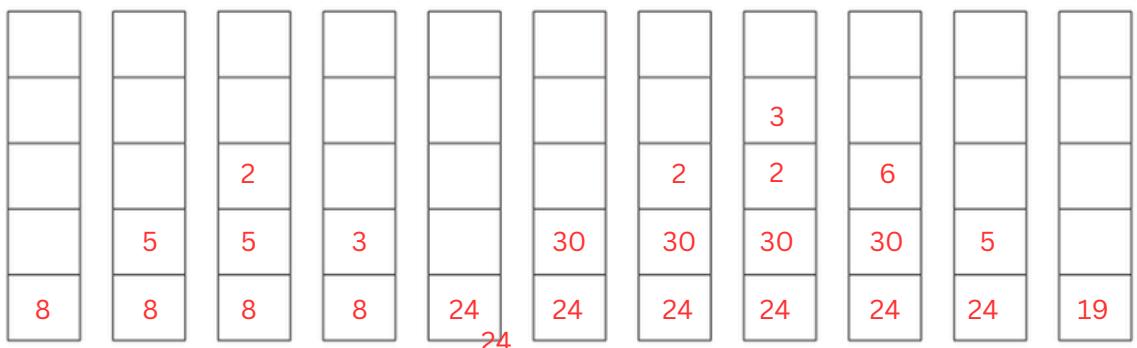
Note : Always Solve Left to Right Whenever operator enters the stack, use the operator on the two values at the top.

(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.



Case 2 : Infix To RPN Conversion

Note : Always use Bodmas rule and left to right is the way you are suppose to convert

1) A + B

A B +

2) A - B * C

A - B C *

A B C * -

3) (P + Q) + (R - P)

PQ+ + RP-

PQ+ RP- +

4) $(a - b) * (a + c) / 7$

$ab - * ac+ / 7$

$ab - ac + * / 7$

$ab - ac + * 7 /$

5) $P + Q - R / S$

$P + Q - RS/$

$PQ+ - RS /$

$PQ + RS / -$

6) $- A / B * 4 / (C - D)$

$A - / B * 4 / CD-$

$A- / B * 4 / CD-$

$A-B/ * 4 / CD-$

$A-B/4* / CD-$

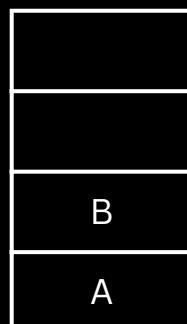
$A-B/4*CD-/$

Case 3 : RPN to Infix Conversion

Note : Use stack concept to solve question try to imagine stacks and the working done in case 2 and always start from left to right

1) $A B -$

$A - B$



2) PQ + M * RP --

P + Q M * RP --

(P+Q) * M RP --

(P + Q) * M R-P -

((P + Q) * M) - (R-P)

3) x w z + y - *

x (w + z) y - *

x (w + z - y) *

x * (w + z - y)

4) b a * c d a + + -

(b * a) c d a + + -

(b * a) c (d + a) + -

(b * a) (c + d + a) -

(b * a) - (c + d + a)

Case 4 : Theoretical Questions

Question : Explain how RPN is used by an interpreter to evaluate expression ?

- Expressions are always evaluated left to right
- Each operator uses the two previous values on the stack (except unary minus)
- If element is a number, push that number onto the stack
- If element is an operator, then pop the first two numbers from the stack
- Perform that operation on those numbers
- Push the result back onto the stack
- End once the last item in the expression has been dealt with.

Question : Advantages of using RPN for the evaluation of an expression ?

- **No need for rules of precedence (BODMAS)**
- **No need for brackets**
- **In RPN evaluation of operators is always left to right**

Question : Identify with reasons, a data structure that could be used to evaluate an expression in RPN ?

- **STRUCTURE: Stack**
- **Reason: Two operands are popped from the stack in the reverse order to how they were pushed.**

Question : Explain why Reverse Polish Notation (RPN) is used to carry out the evaluation of expression ?

- **Reverse Polish Notation provides a clear method of representing an expression**
- **reading from left to right**
- **without the need to use brackets**
- **with no need for rules of precedence / BODMAS**

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.

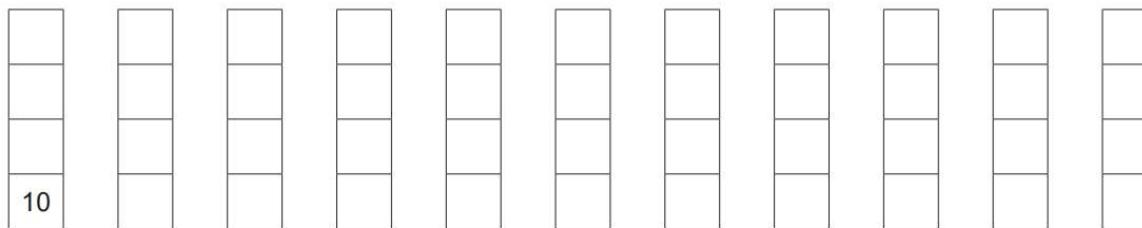
The RPN expression for Y is:

$M\ Q\ /\ P\ Q\ +\ R\ -\ *$

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.

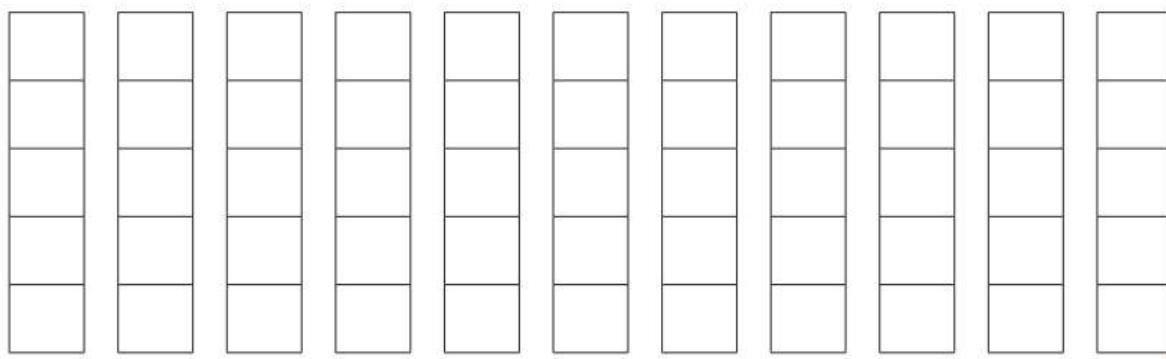
.....
.....
.....
.....
.....
.....
.....
..... [4]

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

8 5 2 - * 30 2 3 * / -

in Reverse Polish Notation (RPN).

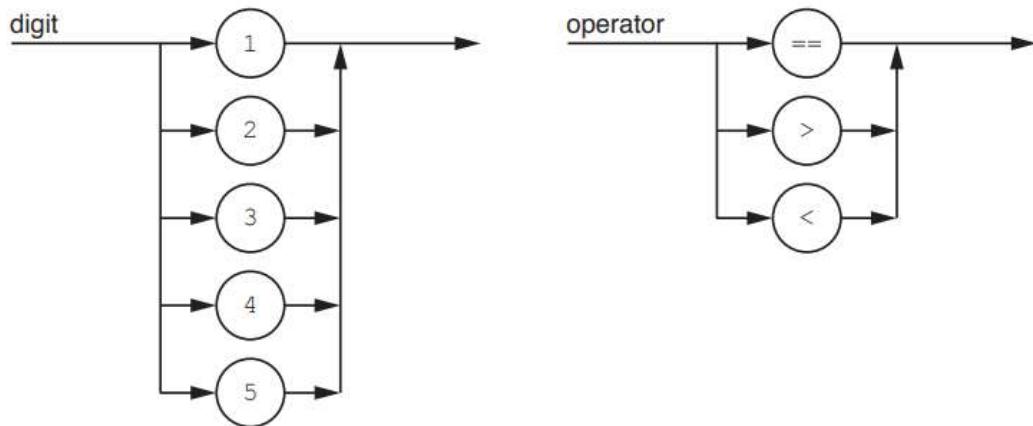
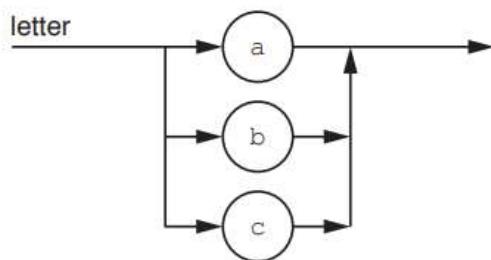
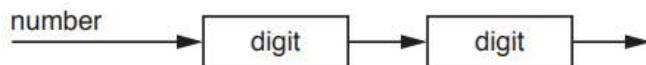
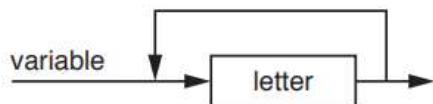
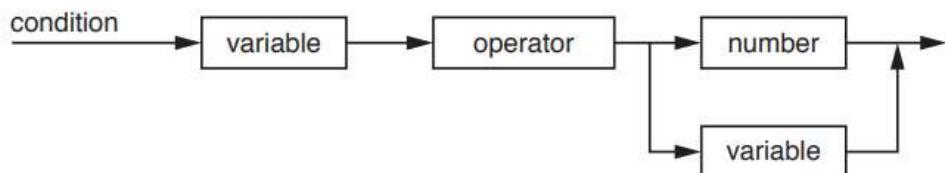
Show the changing contents of the stack as this RPN expression is evaluated.



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.

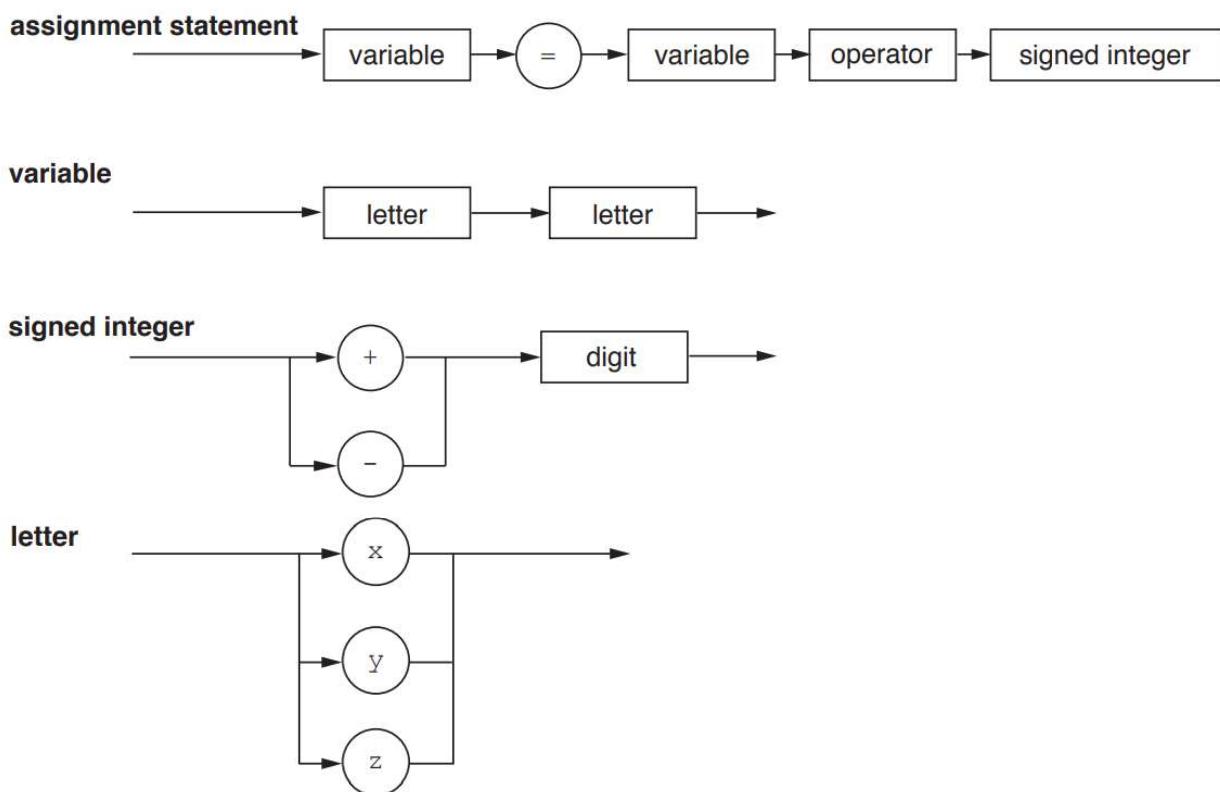
.....
.....
.....
.....

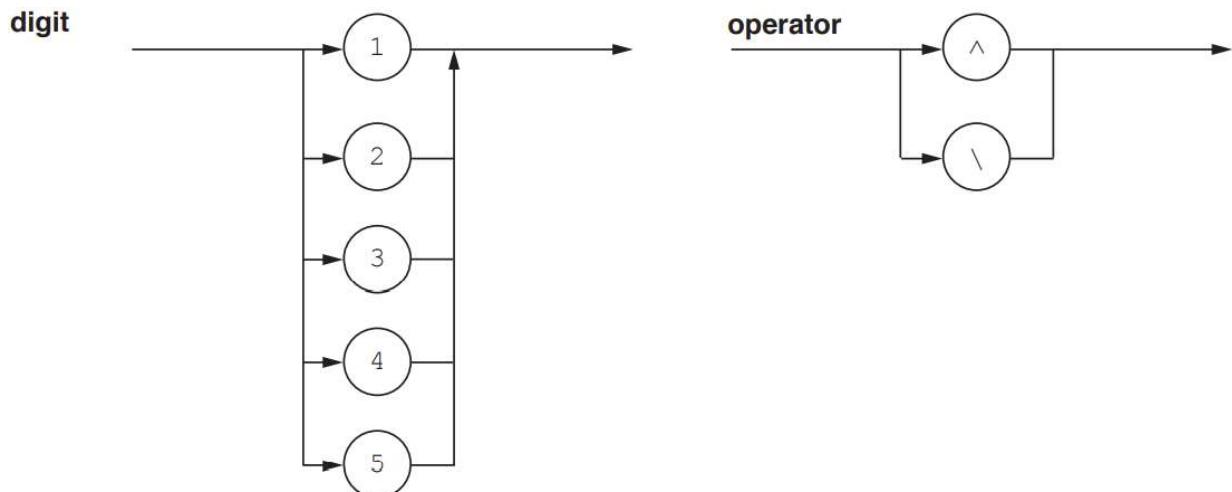
[2]

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





- (a)** The following assignment statements are invalid.

Give the reason in each case.

(i) $xy = xy \wedge c4$

Reason
..... [1]

(ii) $zy = zy \setminus 10$

Reason
..... [1]

(iii) $yy := xz \wedge - 6$

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> ::=

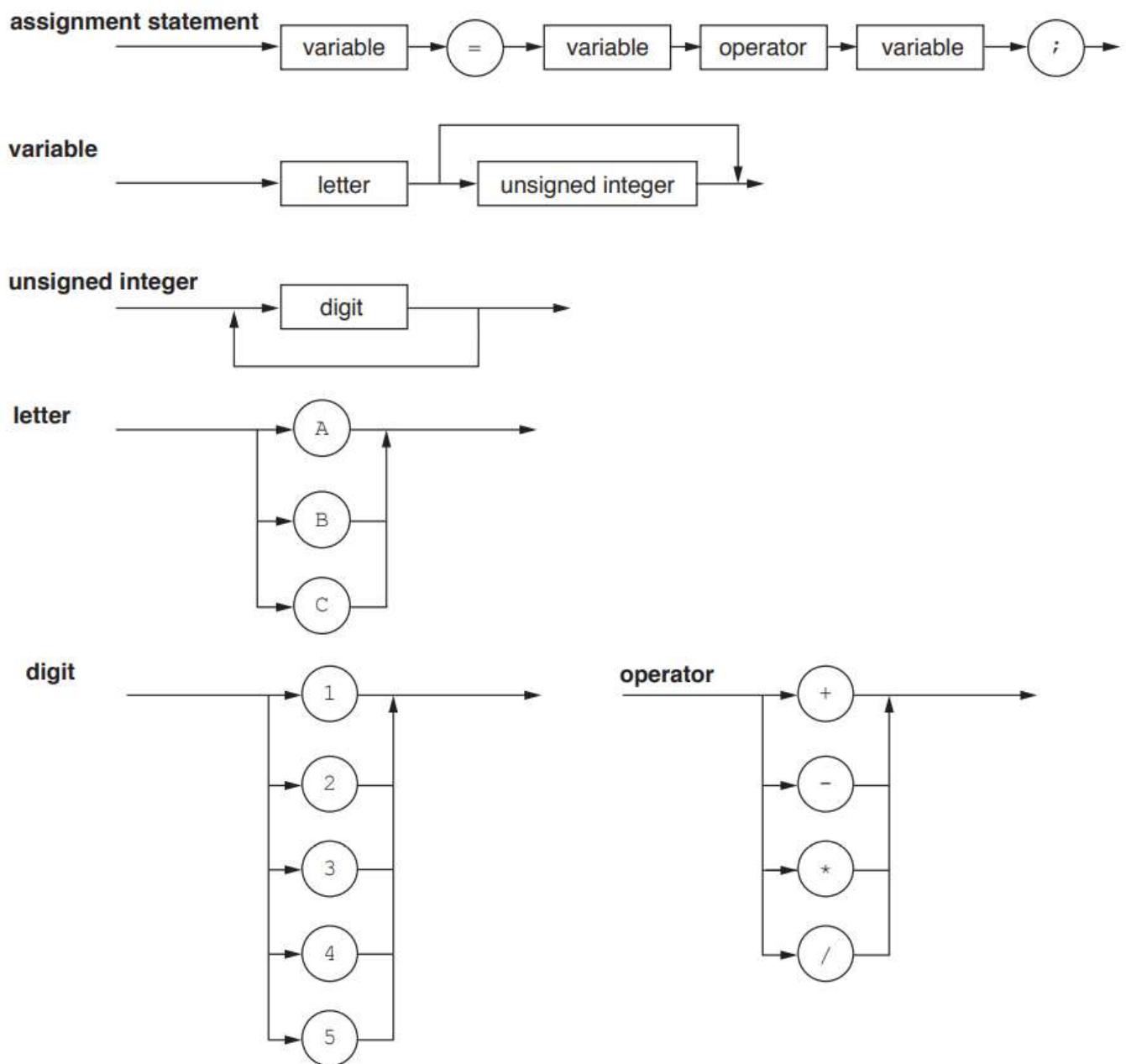
.....

[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: $x \leftarrow A + B + C - D$

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.

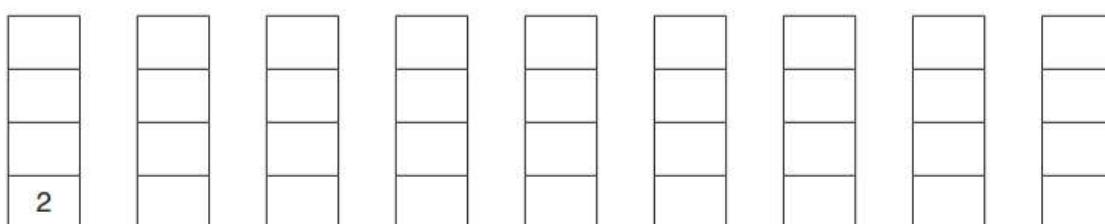
The RPN expression is: b a c + * d + 2 -

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.



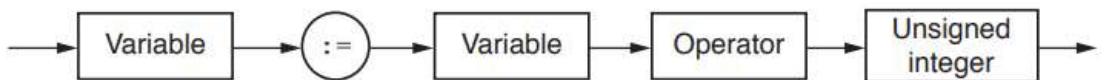
[4]

Question 10

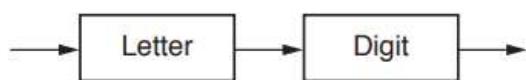
3 The following syntax diagrams for a particular programming language show the syntax of:

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

Assignment statement



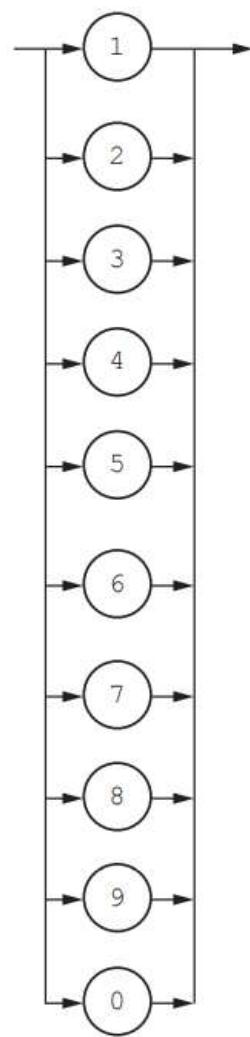
Variable



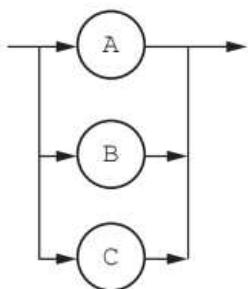
Unsigned integer



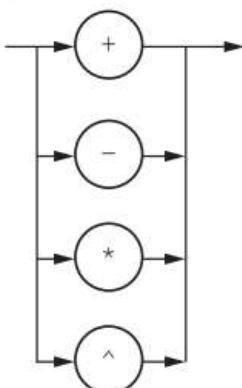
Digit



Letter



Operator



- (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> ::=

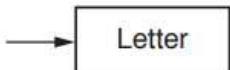
[6]

(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> ::=
.....

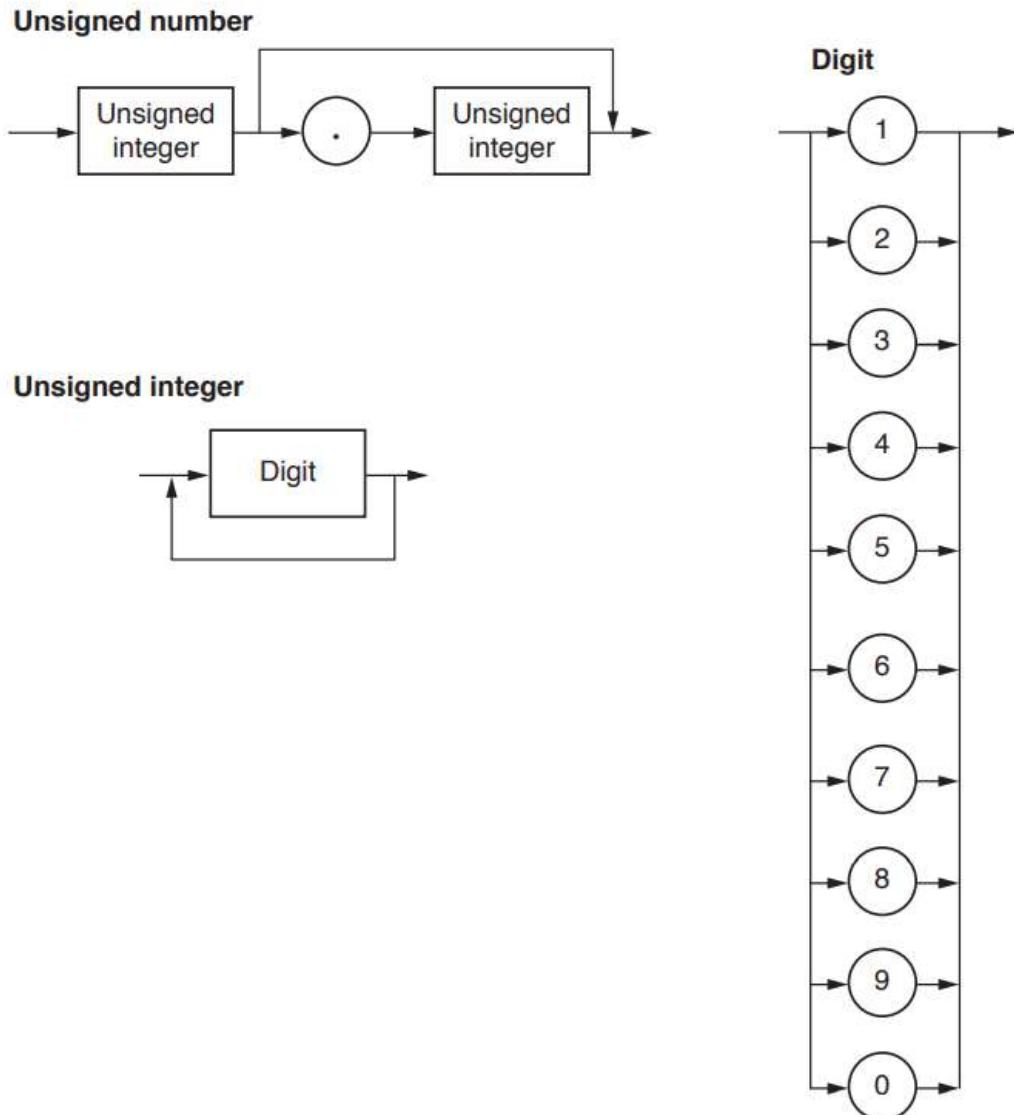
<real> ::=
.....

[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.



(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:

$$x \ w \ z \ + \ y \ - \ *$$

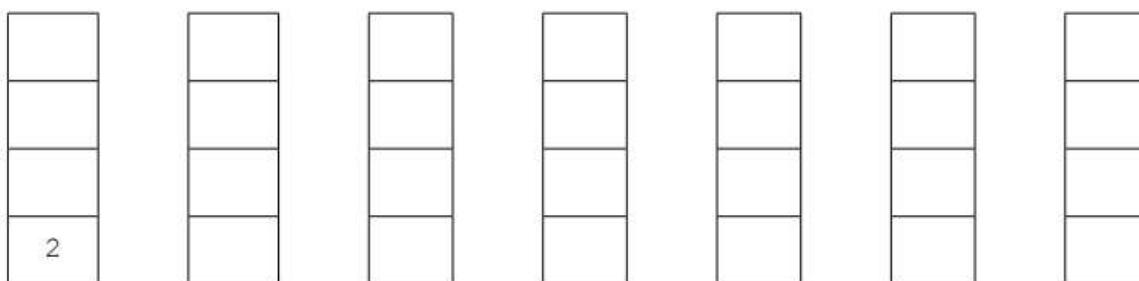
The interpreter evaluates this RPN expression using a stack.

The current values of the variables are:

$$w = 1 \quad x = 2 \quad y = 3 \quad z = 4$$

(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 + + -

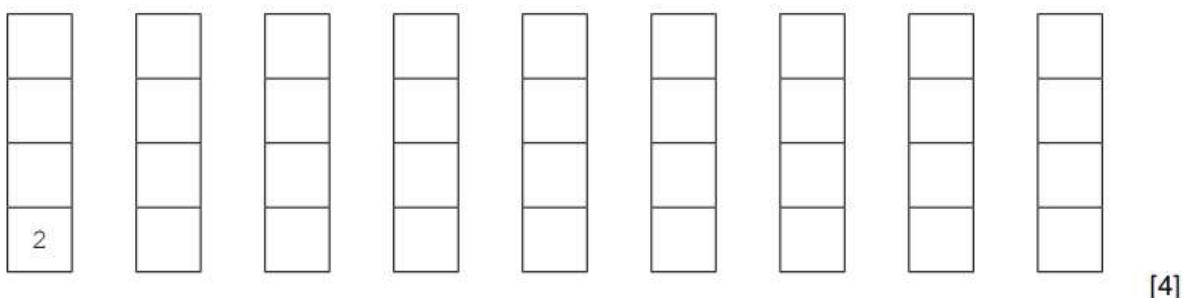
The interpreter evaluates this RPN expression using a stack.

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.



- (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.

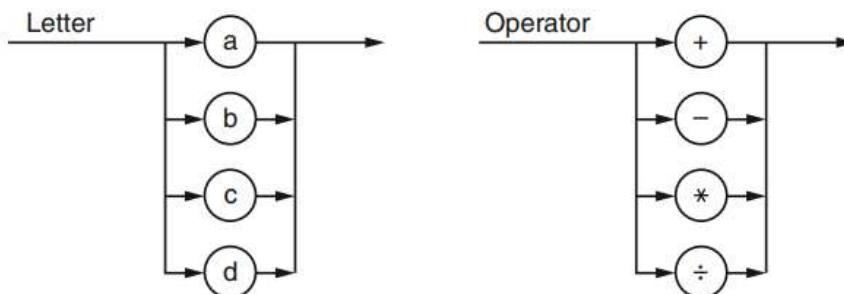
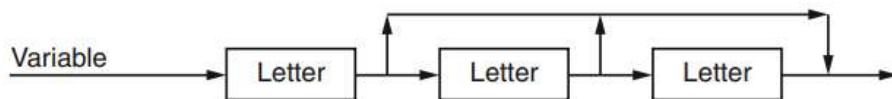
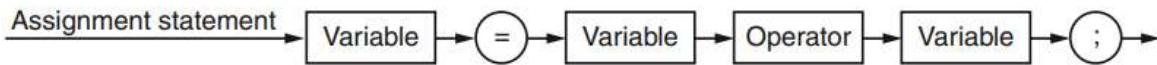
Explain this statement.

.....
.....
.....
..... [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]

..... [1]

(ii) $a = b - 2;$

Reason [1]

..... [1]

(iii) $a = dd * cce;$

Reason [1]

..... [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 \leftarrow B + 2 * 6

.....
.....
.....

[1]

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

X \leftarrow A + B
Y \leftarrow 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]

Question 17

- 4 (a) (i) Explain why Reverse Polish Notation (RPN) is used to carry out the evaluation of expressions.

.....
.....
.....
..... [2]

- (ii) Identify, with reasons, a data structure that could be used to evaluate an expression in RPN.

.....
.....
.....
..... [2]

(b) Write the infix expression in RPN.

$(a - b) * (a + c) / 7$

[1]

(c) Write the RPN expression as an infix expression.

$a b / 4 * a b + -$

[1]

(d) Evaluate the RPN expression:

$a b + c d / /$

where $a = 17$, $b = 3$, $c = 48$ and $d = 12$.

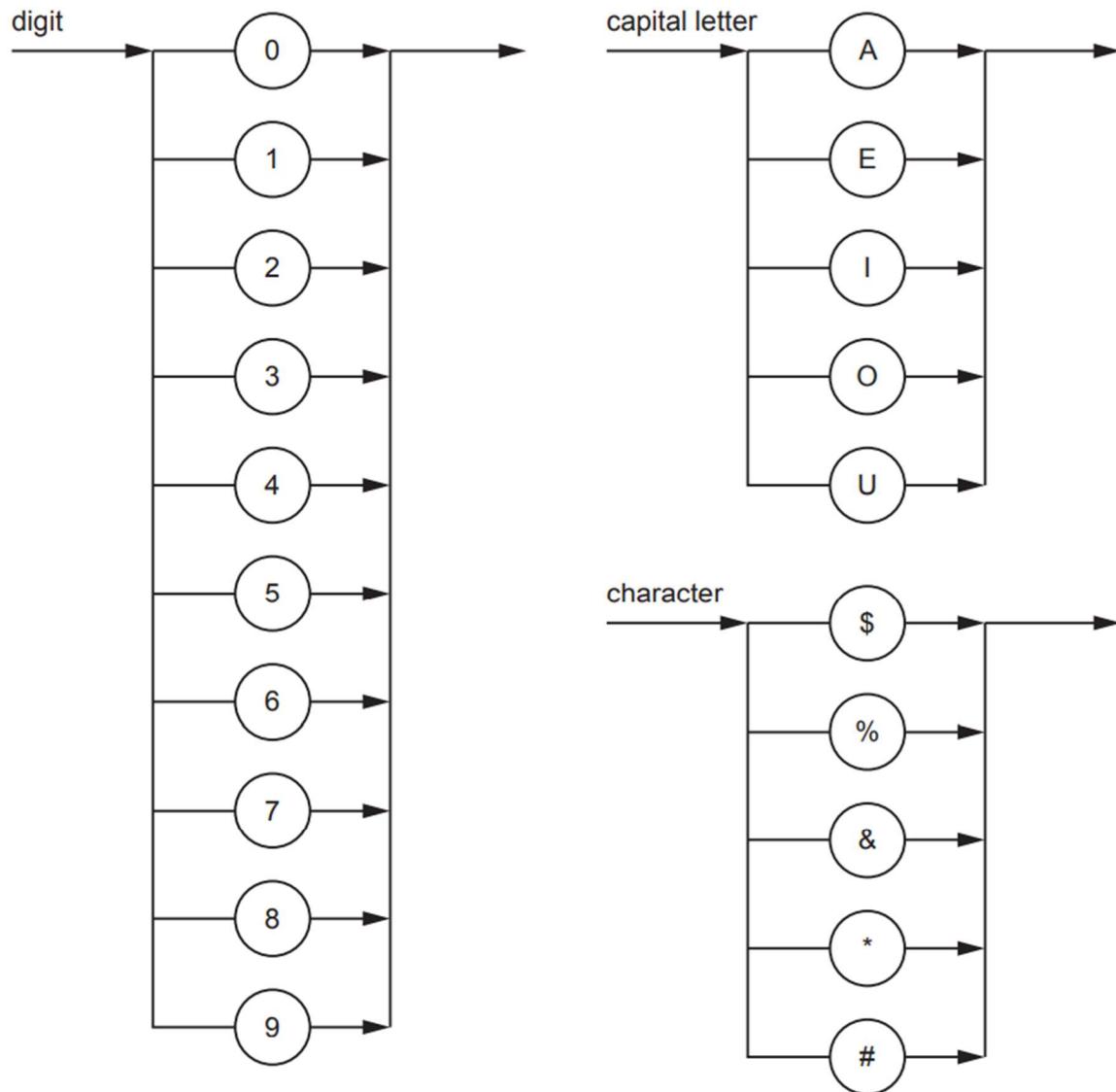
Show your working.

[2]

Question 18

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

- a digit
- a capital letter
- a character.



(a) Write the Backus-Naur Form (BNF) notation of the syntax diagram for character.

.....
.....
.....

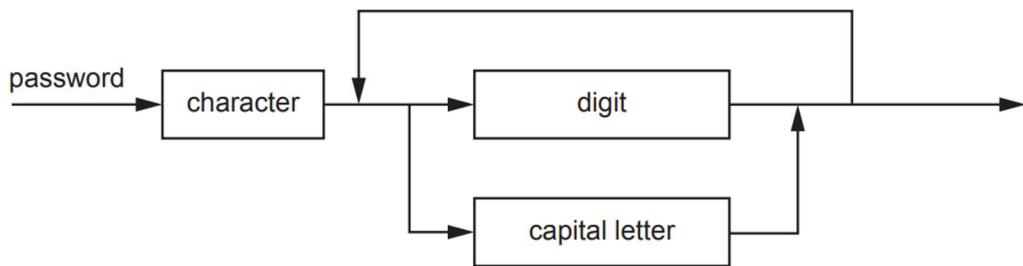
[2]

(b) A password must begin with a character and be followed by one or more digits or capital letters.

(i) State an example of a valid password.

..... [1]

(ii) A valid password is represented by the syntax diagram:



Write the BNF notation of the syntax diagram for password.

.....
.....
.....
.....
.....
.....
.....

[4]

Question 19

5 Part of a program's calculations uses the integer variables j , k , m , n and p .

```
j = 3  
k = 2  
m = 10  
n = (j + k) / (j - k)  
p = m * (m - j * k)
```

(a) Write the Reverse Polish Notation (RPN) for the expression:

$(j + k) / (j - k)$

..... [2]

(b) (i) Show the changing contents of the stack as the value for p is calculated from its RPN expression:

$m\ m\ j\ k\ *\ -\ *$



[4]

(ii) Describe the main steps in the evaluation of this RPN expression using a stack.

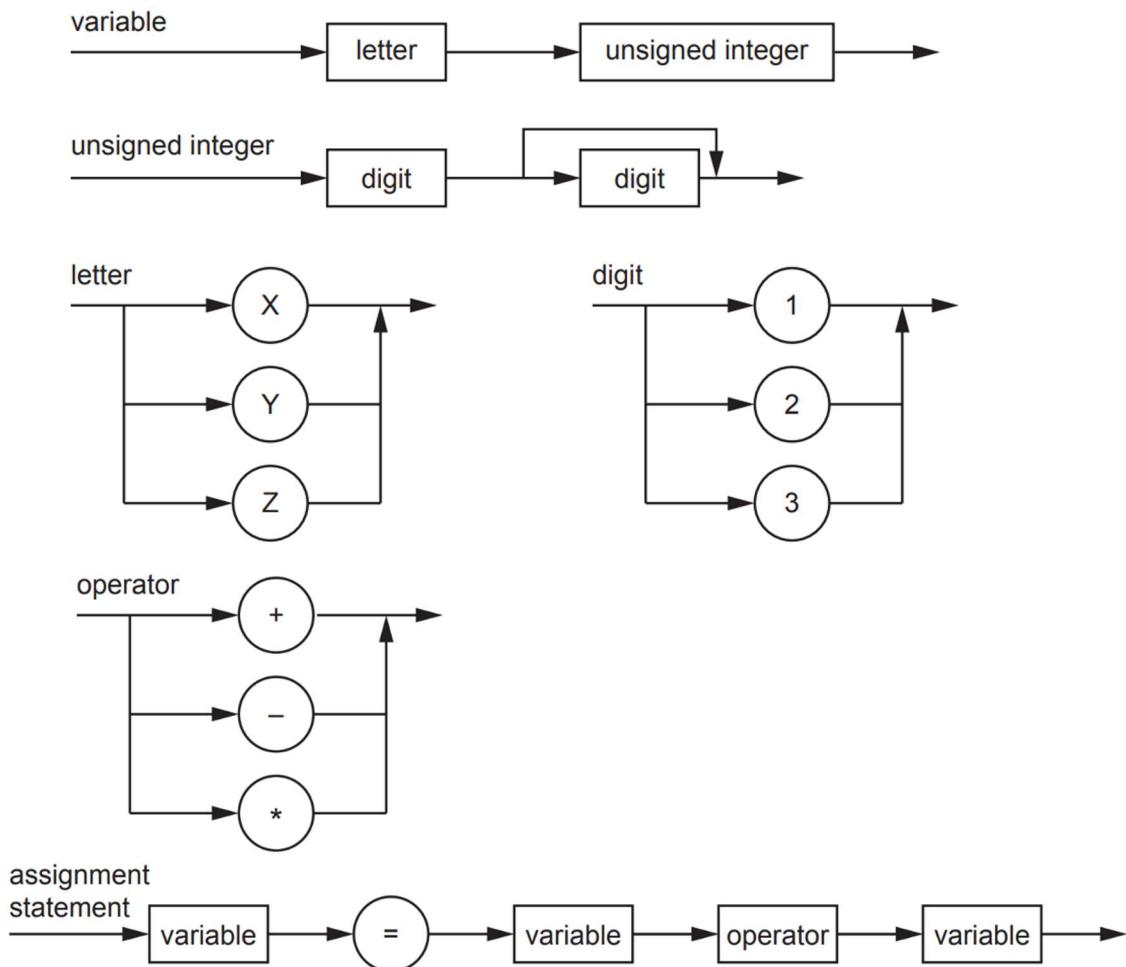
.....
.....
.....
.....
.....
.....
.....
.....
.....

[4]

Question 20

4 The following syntax diagrams show the syntax of:

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



- (a)** The following assignment statements are invalid. State the reason in each case.

X1 = Y2 - 12

Reason
.....

Z = Y12 + Z1

Reason
.....

[2]

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

<letter> has been completed for you.

<variable> ::=

.....
<unsigned_integer> ::=

<letter> ::= X | Y | Z

<digit> ::=

.....
<operator> ::=

.....
<assignment_statement> ::=

[5]

(c) The syntax of an assignment statement is changed to allow each of the variables on the right-hand side of the '=' symbol to be either a variable or an unsigned integer.

(i) Draw a syntax diagram for the new syntax of the **assignment statement**.

[3]

(ii) Write the Backus-Naur Form (BNF) for your syntax diagram.

.....
.....
.....
.....

[3]

Question 21

- 4 Draw **one** line to connect each stage of compilation to its **most appropriate** description.

Stage of compilation	Description
Lexical analysis	minimising a program's execution time and memory requirement
Syntax analysis	converting an intermediate representation of source code into an executable form
Code generation	converting a sequence of characters into a sequence of tokens
Optimisation	directly executing instructions written in a scripting language
	using parsing algorithms to interpret the meaning of a sequence of tokens

[4]

Question 22

- 5 (a) Write the infix expression in Reverse Polish Notation (RPN).

a * b + b - d + 15

..... [1]

- (b) (i) Write the RPN expression in infix form.

a b - c d + * a /

..... [1]

- (ii) Evaluate your infix expression from part (b)(i) when a = 5, b = 10, c = 27 and d = 12.

..... [1]

Question 23

- 2 Lexical analysis and syntax analysis are stages in the compilation of a program.

- (a) Identify **two other** stages that take place during the compilation of a program.

1

.....

2

[2]

- (b) Outline the purpose of syntax analysis.

.....

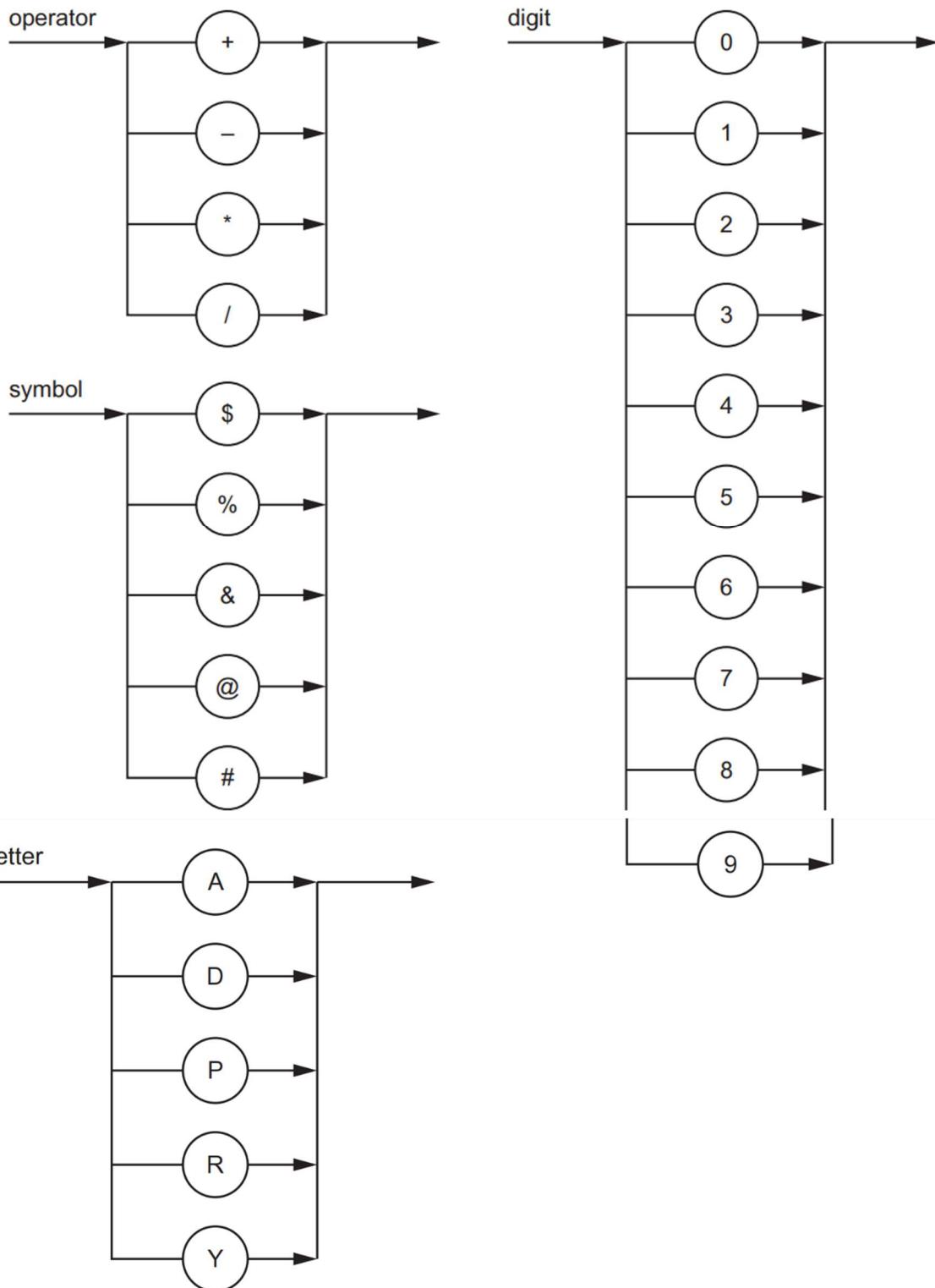
.....

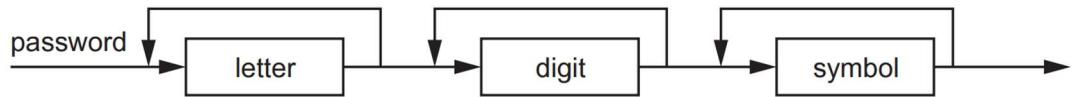
.....

[2]

Question 24

- 6 Several syntax diagrams are shown.





- (a)** State whether each of the following passwords is valid or invalid and give a reason for your choice.

DPAD99\$

Reason

.....

DAD#95

Reason

.....

ADY123?

Reason

.....

[3]

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

<symbol> ::=

.....

<letter> ::=

.....

[1]

- (c) An identifier begins with one or more letters, followed by zero digits or one digit or more digits.

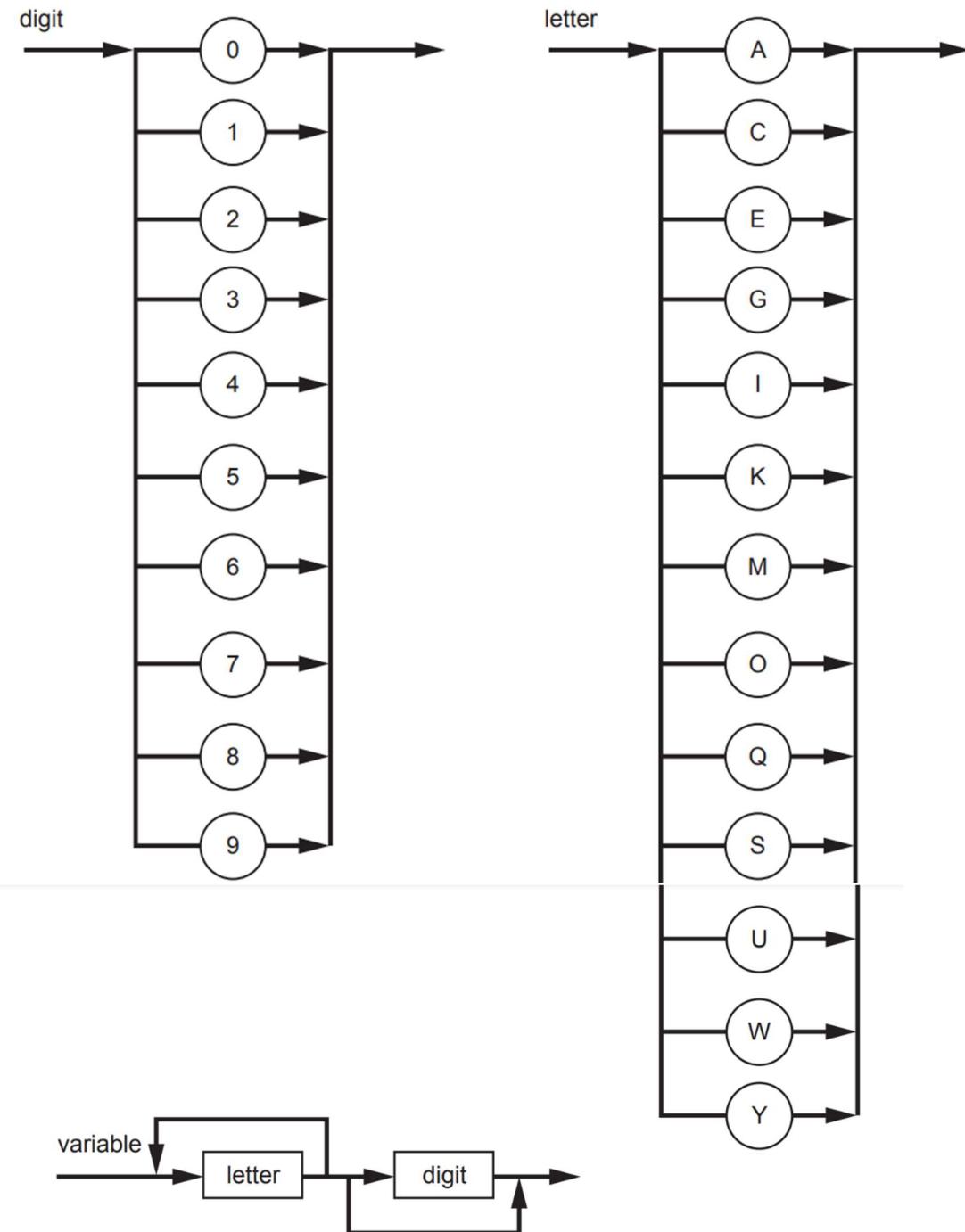
Valid letters and digits are shown in the syntax diagrams on page 6.

Draw a syntax diagram for an identifier.

[4]

Question 25

3 Several syntax diagrams are shown.



- (a)** State whether each variable is valid or invalid **and** give a reason for your choice in each case.

9SW

Reason
.....

UWY

Reason
.....

[2]

- (b)** <word> contains one or more letters.

Complete the Backus-Naur Form (BNF) for <word> **and** use this to complete the BNF for <variable>.

<word> ::=

.....
<variable> ::=

[3]

- (c)** Vehicle registrations must begin with two letters and be followed by one, two or three digits.

Valid letters and digits are shown in the syntax diagrams on page 4.

- (i)** State an example of a valid vehicle registration.

..... [1]

(ii) Draw a syntax diagram for a vehicle registration.

[3]

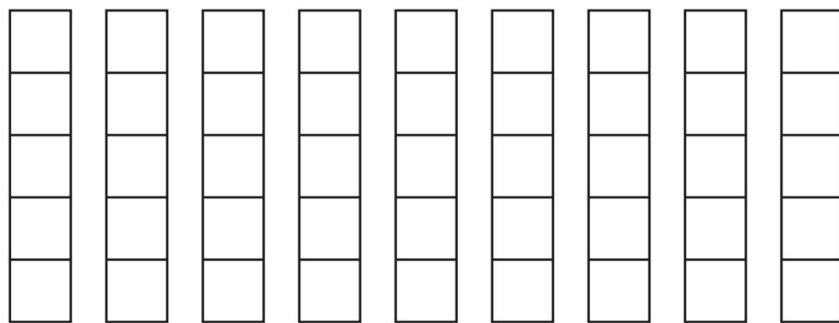
Question 26

- 9 (a) (i) Write the infix expression for this Reverse Polish Notation (RPN) expression:

5 2 - 5 4 + * 9 /

.....
..... [2]

- (ii) Show how the contents of the following stack will change as the RPN expression in part (a)(i) is evaluated.



[4]

- (b) Explain how a stack can be used to evaluate RPN expressions.

.....
.....
.....
.....
.....
.....
.....
..... [3]

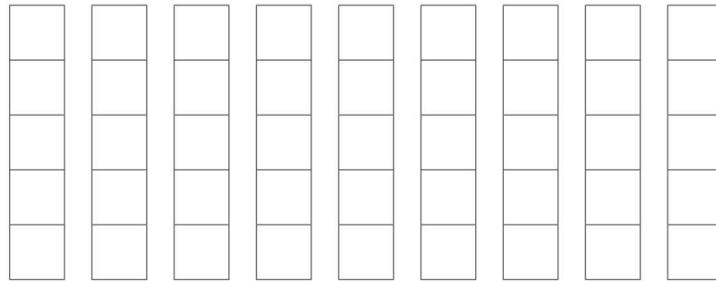
Question 27

- 6 (a) The Reverse Polish Notation (RPN) expression:

a b * 2 / c d / *

is to be evaluated where a = 20, b = 3, c = 10 and d = 5.

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



[4]

- (b) Explain how an expression stored in RPN can be evaluated.

.....
.....
.....
.....
.....
.....
.....
.....
.....

[3]

Question 28

- 5 (a) Write this Reverse Polish Notation (RPN) in infix form:

5 2 + 9 3 - / 3 *

.....
.....
.....
..... [3]

- (b) Write this infix expression in RPN:

((7 + 3) - (2 * 8)) / 6

.....
.....
.....
..... [2]

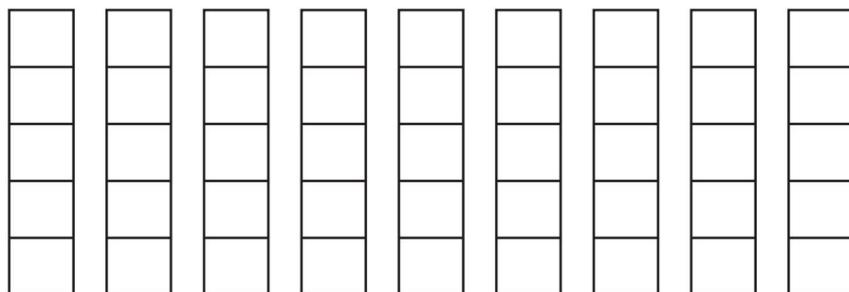
- (c) Evaluate this RPN expression:

a b - c d + * e /

when

a = 17, b = 5, c = 7, d = 3 and e = 10

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



[4]

Question 29

- 5 (a) Write this infix expression in Reverse Polish Notation (RPN):

(7 - 2 + 8) / (9 - 5)

.....
.....
.....
..... [2]

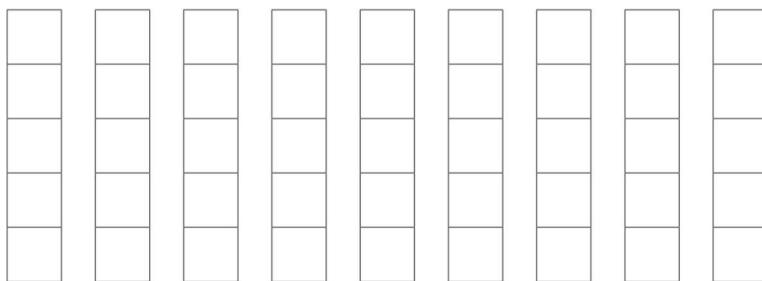
- (b) Evaluate this RPN expression:

a d + a b + c - *

when

a = 6, b = 3, c = 7 and d = 9

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



[4]

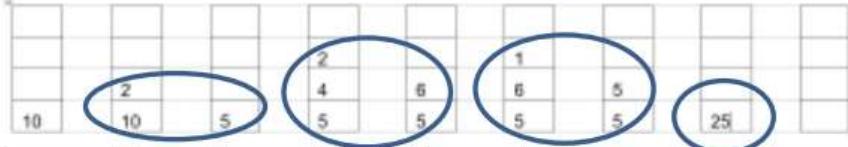
- (c) Write this RPN expression in infix form:

b a c - + d b + * c /

.....
.....
.....
..... [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: <ul style="list-style-type: none"> • 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 <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th rowspan="2" style="width: 20%;">Symbol</th><th colspan="2" style="text-align: center; padding: 5px;">Token</th></tr> <tr> <th style="width: 40%; text-align: center; padding: 5px;">Value</th><th style="width: 40%; text-align: center; padding: 5px;">Type</th></tr> </thead> <tbody> <tr> <td style="padding: 5px;">Counter</td><td style="text-align: center; padding: 5px;">60</td><td style="text-align: center; padding: 5px;">Variable</td></tr> <tr> <td style="padding: 5px;">0</td><td style="text-align: center; padding: 5px;">61</td><td style="text-align: center; padding: 5px;">Constant</td></tr> <tr> <td style="padding: 5px;">Password</td><td style="text-align: center; padding: 5px;">62</td><td style="text-align: center; padding: 5px;">Variable</td></tr> <tr> <td style="padding: 5px;">"Cambridge"</td><td style="text-align: center; padding: 5px;">63</td><td style="text-align: center; padding: 5px;">Constant</td></tr> <tr> <td style="padding: 5px;">1</td><td style="text-align: center; padding: 5px;">64</td><td style="text-align: center; padding: 5px;">Constant</td></tr> </tbody> </table>	Symbol	Token		Value	Type	Counter	60	Variable	0	61	Constant	Password	62	Variable	"Cambridge"	63	Constant	1	64	Constant	3
Symbol	Token																					
	Value	Type																				
Counter	60	Variable																				
0	61	Constant																				
Password	62	Variable																				
"Cambridge"	63	Constant																				
1	64	Constant																				

4(b)	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>60</td><td>01</td></tr> </table> <p>First two cells given in question.</p> <p>1 mark for next 3 cells</p> <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>61</td><td>51</td><td>62</td></tr> </table> <p>1 mark for the remainder</p> <table border="1" style="display: inline-table; vertical-align: middle;"> <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><td>....</td></tr> </table> <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>....</td><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>	60	01	61	51	62	4E	4A	62	04	63	4B	51	62	4C	60	01	60	02	64	4F	62	03	63	52	60	2
60	01																												
61	51	62																											
4E	4A	62	04	63	4B	51	62	4C	60																			
....	01	60	02	64	4F	62	03	63	52	60																			
4(c)(i)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> ∞ Removing the fourth line (LDD 238) ... ∞ Changing operand for second ADD from 236 to 238 ... ∞ ... First three lines and last line unchanged <pre>LDD 236 ADD 237 STO 236 ADD 238 STO 238</pre>	3																											
4(c)((ii))	<p>1 mark per bullet point (max 2)</p> <ul style="list-style-type: none"> ∞ 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)	<p>1 mark per bullet point (max 4)</p> <ul style="list-style-type: none"> ∞ 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)	<p>1 mark per ring (not all stacks are shown) Do not allow operators in stacks Accept intermediate correct stack values</p>	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)	<p>1 mark for each bullet point</p> <pre> <operator> ::= • == > < <number> ::= • <digit><digit> <variable> ::= • <letter> • <letter><variable> <condition> ::= • <variable><operator><number> • <variable><operator><variable></pre>	6

Answer 5

7(a)(i)	<p>1 mark for each bullet point to max 2</p> <p>Keyword table:</p> <ul style="list-style-type: none"> The reserved words used The operators used Their matching tokens 	2
7(a)(ii)	<p>1 mark for each bullet point to max 2</p> <p>Symbol table:</p> <ul style="list-style-type: none"> Identifier name used ... the (data) type ... role, e.g. variable, constant, array, procedure / scope Location (marker) // value of constant 	2
7(a)(iii)	<p>1 mark per bullet point to max 2</p> <ul style="list-style-type: none"> 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)	<p>1 mark per bullet point to max 2</p> <ul style="list-style-type: none"> • 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 <u>signed</u> integer	1
5(a)(ii)	10 is not a valid <u>signed</u> 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)	<p>1 mark per bullet assignment</p> <ul style="list-style-type: none"> ▫ <variable>=<variable><operator><signed integer> ▫ variable ▫ <letter><letter> ▫ signed integer ▫ +<digit> -<digit> ▫ operator ▫ ^ \ <pre> <assignment statement> ::= <variable> = <variable><operator><signed integer> <variable> ::= <letter><letter> <signed integer> ::= +<digit> -<digit> <operator> ::= ^ \ </pre>	4
5(c)	<p>1 mark per bullet</p> <ul style="list-style-type: none"> ▫ <letter> ▫ <letter><variable> <p>For example:</p> <pre> <letter> <letter><variable> <letter> <variable><letter> </pre>	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)	<p>1 mark for each bullet</p> <p>assignment:</p> <ul style="list-style-type: none"> • <variable> = <variable><operator><variable>; <p>variable:</p> <ul style="list-style-type: none"> • <letter> • <letter><unsigned integer> <p>unsigned integer:</p> <ul style="list-style-type: none"> • <digit> • <digit><unsigned integer> <p>operator:</p> <ul style="list-style-type: none"> • + - * / <pre><assignment statement> ::= <variable> = <variable><operator><variable>; <variable> ::= <letter> <letter><unsigned integer> <unsigned integer> ::= <digit> <digit><unsigned integer> <operator> ::= + - * /</pre>	6
2(c)	<p>1 mark per bullet</p> <ul style="list-style-type: none"> • variable with arrow • followed by repeated letter • followed by unsigned integer and arrow <pre> graph LR A[variable] --> B[letter] B --> C[unsigned integer] C --> D[] </pre>	3

Answer 8

6(a)	<p>1 mark for each correct row</p> <table border="1"> <thead> <tr> <th rowspan="2">Symbol</th><th colspan="2">Token</th></tr> <tr> <th>Value</th><th>Type</th></tr> </thead> <tbody> <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> </tbody> </table>	Symbol	Token		Value	Type	Start	60	Variable	1	61	Constant	Number	62	Variable	Counter	63	Variable	12	64	Constant	3
Symbol	Token																					
	Value	Type																				
Start	60	Variable																				
1	61	Constant																				
Number	62	Variable																				
Counter	63	Variable																				
12	64	Constant																				
6(b)	<p>1 mark for each circled section</p>	2																				
6(c)	<p>1 mark per bullet point to max 2:</p> <ul style="list-style-type: none"> ∞ 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 	2																				
6(d)(i)	<p>shortens execution time of program// time taken to execute whole program decreases</p>	1																				
6(d)(ii)	<p>1 mark for each of the following:</p> <ul style="list-style-type: none"> ∞ LDD 236 ADD 237 STO 512 ADD 238 STO 513 ADD 239 STO 514 ∞ Remove line 4 LDD 236 correct lines 3 and 6 in original code ∞ Remove line 5 ADD 237 correct lines 3 and 6 in original code ∞ Remove line 8 and 9 LDD 236 and ADD 237 correct lines 7 and 11 in original code ∞ Remove line 10 ADD 238 correct lines 7 and 11 in original code 	5																				

Answer 9

4(a)	1 mark per row																				
	<table border="1"> <thead> <tr> <th rowspan="2">Symbol</th> <th colspan="2">Token</th> </tr> <tr> <th>Value</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>Number1</td> <td>60</td> <td>Variable</td> </tr> <tr> <td>Number2</td> <td>61</td> <td>Variable</td> </tr> <tr> <td>Answer</td> <td>62</td> <td>Variable</td> </tr> <tr> <td>10</td> <td>63</td> <td>Constant//Literal</td> </tr> <tr> <td>0</td> <td>64</td> <td>Constant//Literal</td> </tr> </tbody> </table>	Symbol	Token		Value	Type	Number1	60	Variable	Number2	61	Variable	Answer	62	Variable	10	63	Constant//Literal	0	64	Constant//Literal
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)	<p>1 mark per bullet point:</p> <ul style="list-style-type: none"> • LDD 236 ADD 237 ADD 238 SUB 239 Copy the instructions STO 235 • 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)	<p>1 mark per bullet point:</p> <ul style="list-style-type: none"> • Code has fewer instructions/occupies less space in memory • shortens execution time of program // time taken to execute whole program decreases 																				
4(d)																					

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)	<pre><assignment_statement> ::= <variable> := <variable> <operator> <unsigned_integer> <variable> ::= <letter> <digit> <unsigned_integer> ::= <digit> <digit> <unsigned_integer> <letter> ::= A B C <operator> ::= + - * ^ }</pre>	1 6
3(c)	<p>Variable</p> <pre> graph LR start(()) --> L1[Letter] L1 --> L2[Letter] L2 --> D1[Digit] D1 --> D2[Digit] D2 --> end(()) style start fill:none,stroke:none style end fill:none,stroke:none style L1 fill:none,stroke:none style L2 fill:none,stroke:none style D1 fill:none,stroke:none style D2 fill:none,stroke:none </pre> <p>Syntax diagram shows one or two letters Syntax diagram shows zero, one or two digits</p>	2
3(d)	<pre><assignment_statement> ::= <variable> := <variable> <operator> <real> <real> ::= <unsigned_integer> . <unsigned_integer></pre>	2

Answer 11

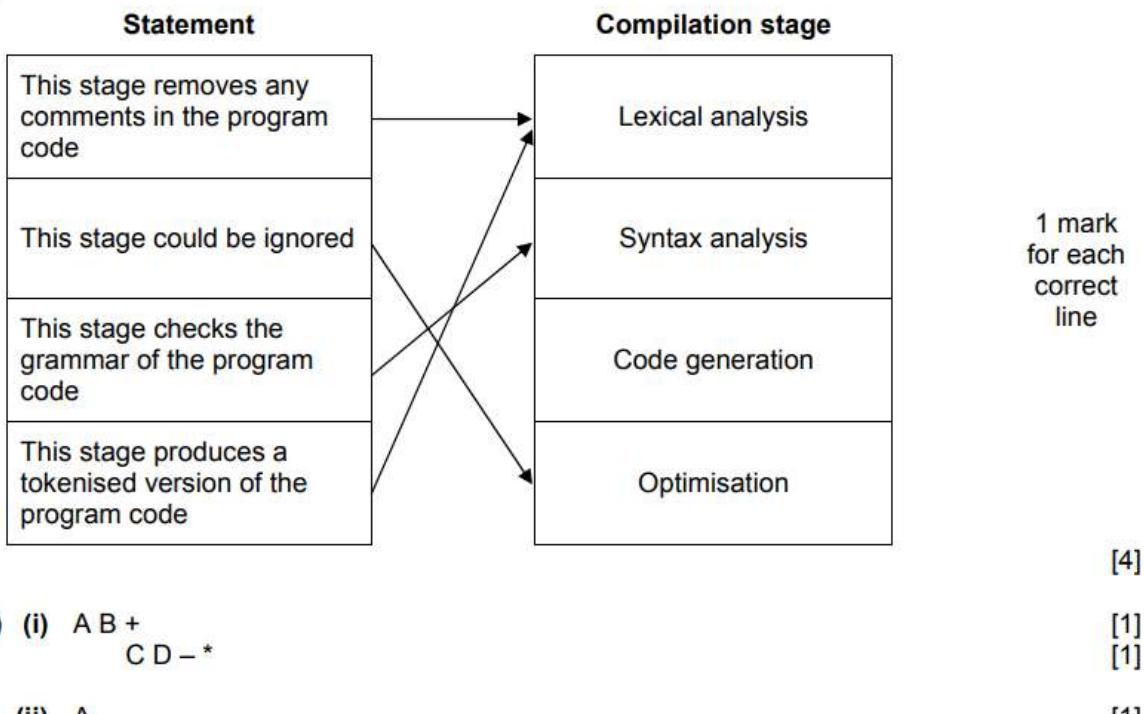
4(a)(i)	Because a valid unsigned integer can be two digits / one or more digits (1) Both 3 and 2 are digits (1)	2
4(a)(ii)	<p>Because a valid unsigned number can be an unsigned integer followed by a decimal point followed by an unsigned integer (1)</p> <p>32 is an unsigned integer and 5 is an unsigned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

<p>4(b)</p> <pre> <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 </pre>	<p>5</p>
<p>4(c)(i)</p> <p>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)</p> <p>Alternative correct answer: MP3 needs both the sign 'box' and the sign diagram for the mark</p>	<p>4</p>

<p>4(c)(ii)</p> <pre> <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) </pre>	<p>4</p>
---	-----------------

Answer 12

2 (a)



(c) (i)

		4		3		
	1	1	5	5	2	
2	2	2	2	2	2	4

+ - *

1
mark
per ring

[4]

(ii) x^*

$(w + z - y)$

Order must be correct for both parts

[1]

[1]

(iii) No need for rules of precedence

[1]

No need for brackets

[1]

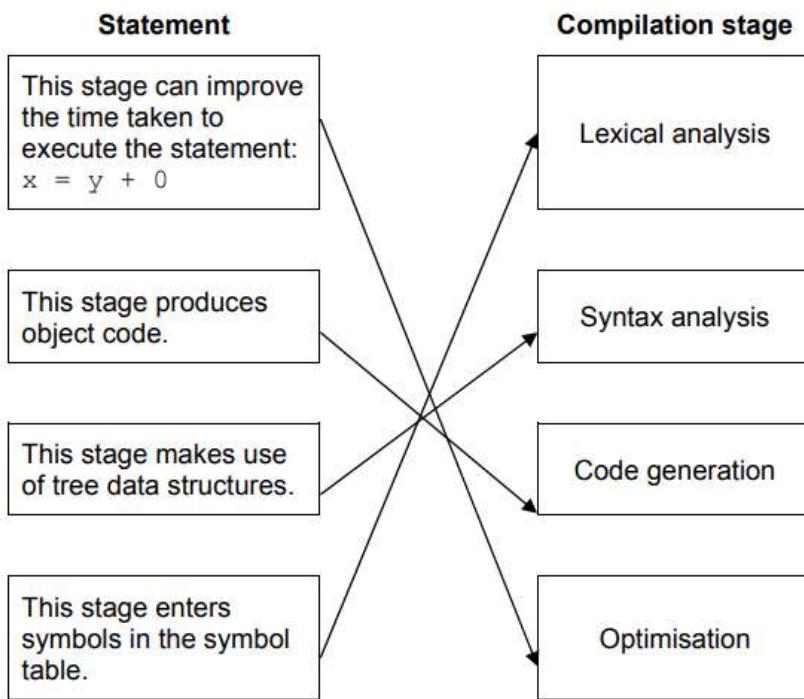
In RPN evaluation of operators is always left to right

[1]

[Max 2]

Answer 13

2 (a)



1
mark for
each
correct
line

[4]

(b) $P Q +$
 $R S / -$

[1]
[1]

(c) (i)

				3	2 3	5		
	2 2 2	2 4	1	1	1 4	1 4	6 4	
*			4	4	+	+	-	

1
mark
per ring

[4]

(ii) $b * a$
 $- (c + d + a)$

[1]
[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)	<pre><assignment statement> ::= <variable> = <variable><operator><variable>; </pre> <pre><variable> ::= <letter> <letter><letter> <letter><letter><letter></pre> <pre><letter> ::= a b c d</pre> <pre><operator> ::= =+ - * ÷</pre>	2 2 1 1

(c)	<u><letter></u> <u><letter><variable></u> // <u><letter></u> <u><variable><letter></u>	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)

6	0	6	5	6	4	6	0	6	4	6	0	6	0	6	4
0	1	1	1	2	A	0	3	2	B	2	1	2	2	3	C

[1+1]

(c) (i) Code optimisation [1]

(ii) LDD 234

[1]

ADD 235

ADD 236

STO 233

[1]

[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
minimises execution time of code // code will execute faster

[1]

[1]

Answer 16

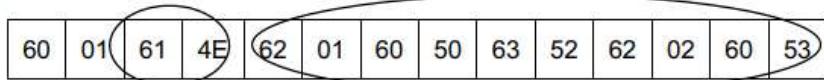
2 (a)

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

[1]

[1+1]

(b)



[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 for each additional instruction; 0 for copy of original code

Answer 17

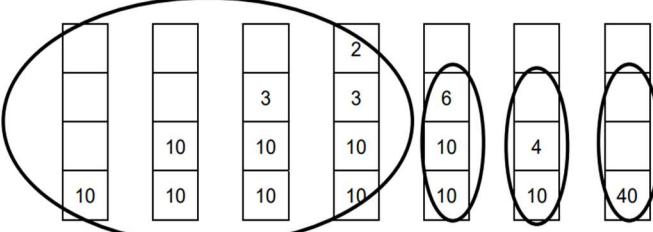
Question	Answer	Marks
4(a)(i)	<p>One mark for each correct marking point (Max 2)</p> <ul style="list-style-type: none"> • Reverse Polish Notation provides an unambiguous method of representing an expression • ... reading from left to right • ...without the need to use brackets • ...with no need for rules of precedence / BODMAS 	2

Question	Answer	Marks
4(a)(ii)	<p>One mark for identification of the data structure, One mark for a sensible reason</p> <p>Either: Structure: stack The operands are popped from the stack in the reverse order to how they were pushed</p> <p>Or: Structure: Binary tree A (binary) tree allows both infix and postfix to be evaluated (tree traversal)</p>	2
4(b)	a b - a c + * 7 /	1
4(c)	a / b * 4 - (a + b)	1
4(d)	<p>1 mark for correct structure 1 mark for correct substitution</p> <p>(a + b) / (c / d) (17 + 3) / (48 / 12)</p>	2

Answer 18

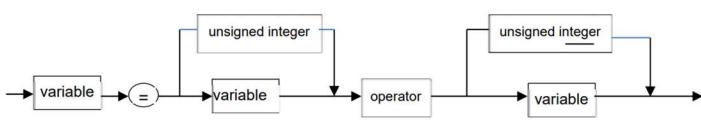
Question	Answer	Marks
4(a)	<p>One mark for each marking point (Max 2)</p> <ul style="list-style-type: none"> • <code><character> ::=</code> • <code>\$ % & * #</code> <p>Complete answer <code><character> ::= \$ % & * #</code></p>	2
4(b)(i)	For example: \$A9E3	1
4(b)(ii)	<p>One mark for each marking point (Max 4)</p> <ul style="list-style-type: none"> • <code><password> ::= <character> ...</code> • <code>... <code></code> • <code><code> ::= ...</code> • <code>... <digit> <capital_letter></code> • <code>... <digit><code> <capital_letter><code></code> <p>Complete answer <code><password> ::= <character><code></code> <code><code> ::= <digit> <capital_letter> <digit><code> <capital_letter><code></code></p>	4

Answer 19

Question	Answer	Marks
5(a)	One mark for each in order $j\ k+\ j\ k-\ /$ $j\ k+$ $j\ k-\ /$	2
5(b)(i)	1 mark per ring Do not allow operators in stacks 	4
5(b)(ii)	Any four from Max 4 Max 3 generic answer only <ul style="list-style-type: none"> • Working from left to right in the expression • PUSH 10/m onto the stack • PUSH the following numbers (10/m, 3/j, 2/k) onto the stack • When the first operator *, is reached • ... POP the top two numbers, 2/k and 3/j • ... apply the operation • PUSH result back onto stack • Continue to the end of the expression 	4

Answer 20

Question	Answer	Marks
4(a)	<ul style="list-style-type: none"> • An unsigned integer, 12, is used instead of the last variable // 12 is not a valid variable • The variable Z is not a valid variable / missing an unsigned integer after the Z 	2
4(b)	One mark per bullet point <ul style="list-style-type: none"> • <code><variable> ::= <letter><unsigned_integer></code> • <code><unsigned_integer> ::= <digit> <digit><digit></code> • <code><digit> ::= 1 2 3</code> • <code>and <operator> ::= + - *</code> • <code><assignment_statement> ::= <variable> =</code> • <code><variable><operator><variable></code> 	5

Question	Answer	Marks
4(c)(i)	One mark adding both boxes... <input type="text"/> unsigned integer <input type="text"/> unsigned integer One mark for correct position(s) and connector(s) ... One mark ... rest correct (assignment statement) 	3

4(c)(ii)	<p>Max three</p> <p>One mark for</p> <pre><assignment_statement> ::= <variable>=</pre> <p>One mark two or three correct options or two marks if all four options correct</p> <pre> <variable><operator><variable> <variable><operator><unsigned_integer> <unsigned_integer><operator><variable> <unsigned_integer><operator><unsigned_integer></pre> <pre><assignment_statement> ::= <variable>= <variable><operator><variable> <variable><operator><variable> <unsigned_integer> <unsigned_integer><operator><variable> <unsigned_integer><operator><unsigned_integer></pre> <p>or</p> <p>One mark for each section</p> <pre><operand> ::= <variable> <unsigned_integer> <assignment_statement> ::= <variable>= <operand><operator><operand></pre> <pre><operand> ::= <variable> <unsigned_integer> <assignment_statement> ::= <variable>=<operand><operator><operand></pre>	3
----------	---	---

Answer 21

Question	Answer		Marks												
4	<p>One mark for each correct line connecting one stage of compilation to a description</p> <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; width: 40%;">Stage of compilation</th> <th style="text-align: center; width: 60%;">Description</th> </tr> </thead> <tbody> <tr> <td style="border: 1px solid black; padding: 5px;">Lexical analysis</td> <td style="border: 1px solid black; padding: 5px;">minimising a program's execution time and memory requirement</td> </tr> <tr> <td style="border: 1px solid black; padding: 5px;">Syntax analysis</td> <td style="border: 1px solid black; padding: 5px;">converting an intermediate representation of source code into an executable form</td> </tr> <tr> <td style="border: 1px solid black; padding: 5px;">Code generation</td> <td style="border: 1px solid black; padding: 5px;">converting a sequence of characters into a sequence of tokens</td> </tr> <tr> <td style="border: 1px solid black; padding: 5px;">Optimisation</td> <td style="border: 1px solid black; padding: 5px;">directly executing instructions written in a scripting language</td> </tr> <tr> <td style="border: 1px solid black; padding: 5px;"></td> <td style="border: 1px solid black; padding: 5px;">using parsing algorithms to interpret the meaning of a sequence of tokens</td> </tr> </tbody> </table>	Stage of compilation	Description	Lexical analysis	minimising a program's execution time and memory requirement	Syntax analysis	converting an intermediate representation of source code into an executable form	Code generation	converting a sequence of characters into a sequence of tokens	Optimisation	directly executing instructions written in a scripting language		using parsing algorithms to interpret the meaning of a sequence of tokens		4
Stage of compilation	Description														
Lexical analysis	minimising a program's execution time and memory requirement														
Syntax analysis	converting an intermediate representation of source code into an executable form														
Code generation	converting a sequence of characters into a sequence of tokens														
Optimisation	directly executing instructions written in a scripting language														
	using parsing algorithms to interpret the meaning of a sequence of tokens														

Answer 22

Question	Answer	Marks
5(a)	$a \ b \ * \ b \ + \ d \ - \ 15 \ +$	1
5(b)(i)	$(a - b) \ * \ (c + d) \ / \ a$	1
5(b)(ii)	-39	1

Answer 23

Question	Answer	Marks
2(a)	One mark per point <ul style="list-style-type: none">• Code generation• Optimisation	2

Question	Answer	Marks
2(b)	One mark per point (Max 2) <ul style="list-style-type: none">• It checks that the code matches the grammar of the language // It checks that the tokens conform with the rules of the programming language• Syntax errors are reported• A parse tree is produced.	2

Answer 24

Question	Answer	Marks
6(a)	<p>One mark per correct valid/invalid and reason combination (Max 3)</p> <p>DPAD99\$ – Valid Reason – 4/multiple letters followed by 2/multiple digits followed by a symbol.</p> <p>DAD#95 – Invalid Reason – The symbol comes before the digits – it should be after.</p> <p>ADY123? – Invalid Reason – The ? is not a valid symbol.</p>	3
6(b)	<p><symbol> ::= \$ % & @ #</p> <p><letter> ::= A D P R Y</p>	1

Question	Answer	Marks
6(c)	<p>One mark per mark point (Max 4)</p> <ul style="list-style-type: none"> • begins with a letter • letter can repeat and digit present • digit can repeat or can be bypassed • correct structure – name, boxes and arrows (in and out). <p>Example answers:</p> <pre> graph LR start(()) --> letter1[letter] letter1 --> digit1[digit] digit1 --> accept1((())) graph LR start(()) --> letter2[letter] letter2 --> letter3[letter] letter3 --> digit2[digit] digit2 --> accept2((())) </pre>	4

Answer 25

Question	Answer	Marks
3(a)	<p>One mark per correct valid/invalid and reason combination (Max 2)</p> <p>9SW – Invalid Reason - This begins with a digit and a variable must begin with a letter</p> <p>UWY – Valid Reason – This begins with a letter and is followed by two other letters.</p>	2
3(b)	<p>One mark per mark point (Max 3)</p> <ul style="list-style-type: none"> • $\langle \text{word} \rangle ::= \langle \text{letter} \rangle$ • $\dots \langle \text{word} \rangle \langle \text{letter} \rangle$ • $\langle \text{variable} \rangle ::= \langle \text{word} \rangle \langle \text{word} \rangle \langle \text{digit} \rangle$ <p>Example answers</p> <p>$\langle \text{word} \rangle ::= \langle \text{letter} \rangle \langle \text{word} \rangle \langle \text{letter} \rangle$ $\langle \text{word} \rangle ::= \langle \text{letter} \rangle \langle \text{word} \rangle \langle \text{letter} \rangle$</p> <p>$\langle \text{variable} \rangle ::= \langle \text{word} \rangle \langle \text{word} \rangle \langle \text{digit} \rangle$ $\langle \text{variable} \rangle ::= \langle \text{word} \rangle \langle \text{digit} \rangle \langle \text{word} \rangle$</p>	3
3(c)(i)	<p>Answer must be two letters followed by one, two or three digits using the letters and digits on the syntax diagram.</p> <p>Example answer AC768</p>	1
3(c)(ii)	<p>One mark per mark point (Max 3)</p> <ul style="list-style-type: none"> • always has only two letters • one, two or three digits possible • correct arrows, boxes and name of syntax diagram <p>Example answer</p> <p>Vehicle registration</p> <pre> graph LR A[letter] --> B[letter] B --> C[digit] C --> D[digit] D --> E[digit] E --> F[digit] E --> G[digit] </pre>	3

Answer 26

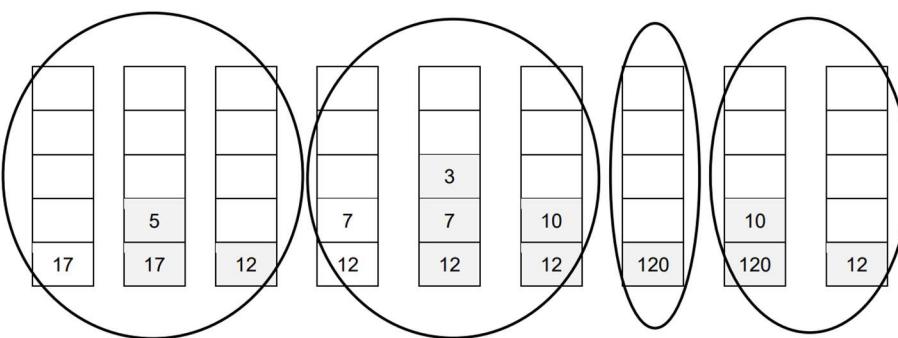
Question	Answer	Marks
9(a)(i)	<p>One mark per mark point (Max 2)</p> <ul style="list-style-type: none"> • $(5 - 2)$ • $\star (5 + 4) / 9$ <p>Final correct expression $(5 - 2) \star (5 + 4) / 9$</p>	2
9(a)(ii)	<p>One mark per ring (Max 4)</p> <p>OR</p>	4
9(b)	<p>One mark per mark point (Max 3)</p> <p>MP1 Evaluate the RPN expression from left to right MP2 Push each element of the RPN expression onto the stack in order until an operator is reached MP3 Pop the last two elements from the stack and apply the operator MP4 Push the result of the operation onto the stack MP5 Repeat the process until the whole expression is evaluated.</p>	3

Answer 27

Question	Answer	Marks
6(a)	<p>One mark per ring (Max 4).</p> <p>OR</p>	4
6(b)	<p>One mark per mark point (Max 3)</p> <p>MP1 The (RPN) expression is read from left to right, one item at a time</p> <p>MP2 Each element is checked to see if it is an operator or a value</p> <p>MP3 Values are pushed onto a stack until an operator is found</p> <p>MP4 The operator is applied to the last two values on the stack and the result is pushed back onto the stack</p> <p>MP5 This repeats until a single value remains, which is the solution.</p>	3

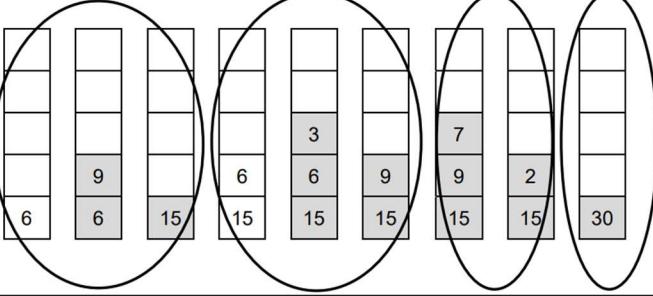
Answer 28

Question	Answer	Marks
5(a)	One mark per correct term (Max 3) $(5 + 2)$ $/ (9 - 3)$ $* 3$ Complete correct answer $((5 + 2) / (9 - 3)) * 3$	3
5(b)	One mark 7 3 + One mark 2 8 * - 6 / Complete answer $7 3 + 2 8 * - 6 /$	2

Question	Answer	Marks
5(c)	One mark per ring (Max 4) 	4

Answer 29

Question	Answer	Marks
5(a)	One mark $7 - 2 +$ One mark $9 - 5 /$ Complete answer $7 - 2 + 9 - 5 /$	2

Question	Answer	Marks
5(b)	One mark per ring (Max 4) 	4
5(c)	One mark per correct term (Max 3) $(a - c + b)$ $* (d + b)$ $/c$ Complete correct answer $((a - c + b) * (d + b)) / c$ Or $(a - c + b) * (d + b) / c$	3