3.4.3 Translation Software

May/June 2003.P3

1. (b) Using examples, explain what happens during the lexical analysis stage of compilation. **[4]**

Oct/NOV 2003.P3

3. (b) Describe the process of syntax analysis when compiling a program. **[3]**

May/June 2004.P3

4. Describe what happens during the syntax analysis stage of compilation. **[5]**

7. VARIABLE NAME is defined in a particular language as an alphabetic character which may be followed by two digits or another alphabetic character.
Given that, in Backus-Naur Form (BNF), an alphabetic character is called an ALPHA and is defined as

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

and a digit is defined as

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

(a) Use BNF and the above definitions (that do not need to be written out again), to define <VARIABLE NAME> [4]

(b) The definition of a variable name is altered.
A variable name is now defined as either

• an alpha followed by two digits, where the first digit must not be zero,

OR

• an unlimited set of alpha characters.

Write new rules in BNF that will define the new <VARIABLE NAME>. **[4]**

May/June 2005.P3

6. (b) Explain what happens during the lexical analysis stage of compilation. **[5]**

(c) Describe two things that happen during code generation. **[4]**

May/June 2006.P3

1 (c) Explain how errors in the

(i) reserved words,
(ii) variables

used in high level language instructions are recognised by the translator program. **[4]**

Oct/NOV 2006 .P3

4. (a) Explain the purpose of the code generation phase of compilation, including the principle of optimisation. **[3]**

Oct/NOV 2007.P3

7. Describe how a variable in a high level language program is dealt with by a compiler. **[6]**

10. A variable name is defined in a particular system as:

- one or two letters, followed by
- any number of digits(including zero) followed by either a
  - $ sign if there are no digits,
  - & sign if there are any digits.

Draw a syntax diagram which describes a variable name. **[6]**

May/June 2008.P3

9. (a) Describe how a compiler recognises a syntax error. **[4]**

(b) Describe the code generation phase of compilation. **[4]**

Oct/NOV 2008.P3

10. The following rules define <WORD> in a piece of text.

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

(i) State why `Hello` is not a word. **[1]**

(ii) `<SENTENCE>` is a set of words ending with a full stop (`.`) or a question mark (`?`)
Define `<SENTENCE>`.
(There is no need to rewrite the rules for `<LETTER>` and `<WORD>`). **[5]**

Oct/NOV 2009. P31

6. Two of the stages which a high level language program undergoes during compilation are lexical analysis and syntax analysis.

Discuss how errors are discovered during each of these two stages. **[5]**

3.4.3 Translation Software

May/June 2010. P32

1. (a) Explain the differences between using a compiler and an interpreter for:

(i) the translation of a high-level language program,
(ii) the execution of a high-level language program. **[6]**

(b) One phase of compilation is the code generation phase.

(i) Name the two phases that are carried out before code generation. **[2]**
(ii) Describe the code generation phase. **[3]**

10 An `INTEGER` is defined in a certain language by the following BNF rules:

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

(a) Explain why each of the following is not an integer:

(i) `.037` **[1]**
(ii) `037` **[1]**

(b) A REAL is defined as an integer or a zero followed by a . (point) followed by an unlimited number of digits (including none).
Produce BNF rules to define `REAL`. (The BNF rules stated above do not need to be reproduced.) **[3]**

(c) Draw a syntax diagram of REAL. You may use `NZDIGIT`, `DIGIT`, `POINT` and `ZERO` in your diagram but no other values. **[4]**

May/June 2010. P33

1. (a) Explain the differences between the lexical analysis stage and the syntax analysis stage in the compilation of a high level language program. **[6]**

(b) One phase of compilation is the code generation phase.

Describe the code generation phase. **[3]**

10 A VARIABLE is defined in a certain language by the following BNF rules:

```
<VARIABLE>::=<GROUP><IDENTIFIER>|<GROUP>
<GROUP>::=<LETTER>|<LETTER><DIGIT>|<LETTER><GROUP>
<LETTER>::= A|B|C
```

3.4.3 Translation Software

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

(a) Explain why each of the following is not a variable:

(i) D2% **[1]**
(ii) $C2 **[1]**

(b) MAIN_VARIABLE is defined as a GROUP which has a NZDIGIT (a DIGIT which must not be a zero) as the first character and cannot include an IDENTIFIER, but has a ! or a & as the last character.

For example 1A2! and 9BC& are examples of MAIN_VARIABLE but 0A2! and 9BC%& are not examples of MAIN_VARIABLE.

Produce a set of rules to define MAIN_VARIABLE (The BNF rules stated above do not need to be reproduced). **[3]**

(c) Draw a syntax diagram of VARIABLE. You may use DIGIT, LETTER and IDENTIFIER in your diagram but no other values. **[4]**

Oct/NOV 2010. P31/P32

4. (a) Explain differences between using an interpreter and a compiler when translating and executing a source code program. **[6]**

(b) Explain the lexical analysis phase of compilation. **[5]**

Oct/NOV 2010. P33

4. (b) Describe the process of syntax analysis when compiling a high-level language program. **[3]**

(c) Describe two things that happen during the code generation stage of compilation. **[4]**

May/June 2011. P31/P32

9. (a) Explain the need for reverse Polish notation. **[2]**
(b) Show, with the aid of diagrams, how a stack is used to turn the reverse Polish Expression
ab+cde-*-
into an expression in infix notation. **[6]**

May/June 2011. P33

9 (a) Explain why reverse Polish notation is used in computer processing. **[2]**

(b) (i) Show how the following infix expression can be represented as a binary tree.

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

(ii) Use the tree to write down the reverse Polish form of the expression. **[6]**

11. (a) Describe the differences between interpretation and compilation of a high-level language program. **[3]**

Oct/NOV 2011. P31

9 (a) (i) Describe what happens during the lexical analysis phase of compilation. **[4]**

(ii) Explain how syntax errors are identified during compilation. **[3]**

10 A variable identifier in a certain programming language is defined in BNF (Backus-Naur form) as:

```
<non-zero-digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<digit> ::= 0 | <non-zero-digit>
<letter> ::= A | B | C | x | y | z
<group> ::= <letter> | <letter><group>
<variable-identifier> ::= <non-zero-digit><group><digit> | <non-zero-
digit><group>
```

(a) Explain why each of the following variable identifiers is invalid:

(i) `23A`
(ii) `2X`
(iii) `2ACB24` **[3]**

(b) Using only the terms:

- non-zero-digit
- digit
- letter
- variable-identifier

draw a syntax diagram to show the definition of a variable identifier. **[4]**

Oct/NOV 2011. P32

9 (a) (i) Describe what happens during the syntax analysis phase of compilation. **[4]**

(ii) Explain how syntax errors are identified during compilation. **[3]**

10 A variable identifier in a certain programming language is defined in BNF (Backus-Naur form) as:

```
<non-zero-digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<digit> ::= 0 | <non-zero-digit>
<letter> ::= A | B | C | x | y | z
```

```
<group> ::= <letter> | <letter><group>
<variable-identifier> ::= <digit><group><non-zero-digit> | <digit><group>
```

(a) Explain why each of the following variable identifiers is invalid:

(i) `0A0`
(ii) `2WA`
(iii) `2ACB24` **[3]**

(b) `5Ay6` can be expressed as `<digit><group><non-zero-digit>`.
Explain why `5Ay6` is a valid variable identifier. **[4]**

Oct/NOV 2011. P33

7 (a) Explain how variables are managed during the different stages of compilation of a high-level language program.

[7]

10 Workers in a factory each have an identity code which identifies their record in the computer system.

The identity code contains letters and numbers and is defined using BNF (Backus-Naur form) as:

```
<identity-code> ::= <group> | <group><number>
<group> ::= <letter> | <letter><group>
<letter> ::= A | B | C | D
<number> ::= <digit> | <digit><digit>
<digit> ::= 0 | 1 | 2
```

(a) Explain why each of the following identity codes is invalid:

(i) `2BA`
(ii) `XAA`
(iii) `ACB021` **[3]**

(b) The definition is changed to allow only a number which begins with a 1 or a 2. The first digit in the number is now defined as

```
<non-zero-digit> ::= 1 | 2
```

Draw a syntax diagram to show the definition of an identity code using only the terms:

- non-zero-digit
- digit
- letter
- identity-code **[4]**

May/June 2012. P31/32

3.4.3 Translation Software

4 Expressions can be written in either infix or reverse Polish notation.

(a) Evaluate this reverse Polish expression:

```
4 6 * 3 -
```

[1]

(b) Write the following infix expressions in reverse Polish.

(i) `(a-5)/(b+c)` **[1]**

(ii) `2 * 3 + 6 / 2` **[2]**

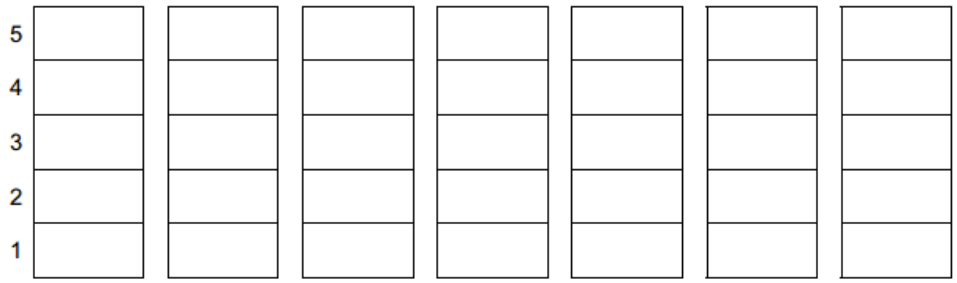
(c) Describe one benefit of storing an expression in reverse Polish. **[1]**

(d) An expression in reverse Polish can be evaluated on a computer system using a stack.

(i) Describe the operation of a stack. **[1]**

(ii) A stack is to be implemented as an array with an integer variable to point to the 'top of stack' index position.
State whether this is a static data structure or a dynamic data structure and explain your choice. **[2]**

(iii) The reverse Polish expression `3 7 * 6 + 9 /` is to be evaluated using a stack.
The first available location on the stack is 1.
Show how the contents of the stack change as this expression is evaluated.



[4]

Oct/NOV 2012. P32

4 (c) Explain what is meant by code optimisation. **[3]**

Oct/NOV 2012. P33

4 Two types of software which are used to translate high-level programs are a compiler and an interpreter.

(c) Describe what happens during the syntax analysis stage of translation. **[3]**

May/June 2013. P31/32

2 (a) Explain the use of Backus-Naur Form (BNF) in computer science. **[2]**

(b) A set of BNF rules are defined as follows:

3.4.3 Translation Software

1. `<BinaryDigit> ::= 0 | 1`
2. `<Parentheses> ::= "`
3. `<Binary> ::= <BinaryDigit> | <BinaryDigit> <Binary>`
4. `<BinaryString> ::= <Parentheses> <Binary> <Parentheses>`

(i) A BNF rule can be recursive.
Explain what is meant by recursive. **[1]**

(ii) State the rule above which is recursive. **[1]**

(iii) For each expression state whether it represents a valid or invalid binary string. List the rule number(s) in the order you have applied them to arrive at your decision.

Expression	Valid/Invalid	Rules used
0		
"1"		
"001"		

[7]

(c) The rules used in (b) are to be extended to allow a binary string to start with a $ character.
For example "$010" is a valid binary string.
Rewrite the set of rules to allow this additional format. **[2]**

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

```
// program written 12 June 2013
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) Describe what information is contained in these tables. **[2]**

(ii) Explain how the table contents are used to translate the source code. **[2]**

(iii) Describe one other task done at the lexical analysis stage which does not involve the use of the keyword table or symbol table. **[1]**

(b) The final stage of compilation is code optimisation.

(i) Explain what is meant by code optimisation. **[2]**

(ii) Give one practical example of code which would benefit from optimising. **[1]**

May/June 2013. P33

2 (a) Explain the need for Backus-Naur Form (BNF) in computer science. [2]

(b) A set of BNF rules describe a data structure called a list.

```
1. <Char> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
2. <ListItem> ::= <Char>
3. <Comma> ::= ,
4. <LSquareBracket> ::= [
5. <RSquareBracket> ::= ]
6. <Contents> ::= <ListItem> | <ListItem> <Comma> <Contents>
7. <List> ::= <LSquareBracket> <Contents> <RSquareBracket>
```

(i) A BNF rule can be recursive.
Explain what is meant by recursive. **[1]**

(ii) State the rule above which is recursive. **[1]**

(iii) For each expression state whether it represents a valid or invalid list. State the rule number(s) in the order you have applied them to arrive at your decision.

Expression	Valid/Invalid	Rules used
[g]		
[dc]		
[w,a]		

[7]

(c) The rules used in (b) are to be extended to allow any one list item to be one or two characters.
For example, the following will both be valid lists:
`[a,ng] [fg,jk,mn]`
Write the new and/or amended BNF rule(s) which are required to include two character items. **[3]**

Oct/Nov 2013.P31

1 (a) Convert the following infix expressions into reverse Polish notation:

(i) `(a + b) / 7` **[1]**

3.4.3 Translation Software

(ii) `2 / (3 * Z + 5)` **[2]**

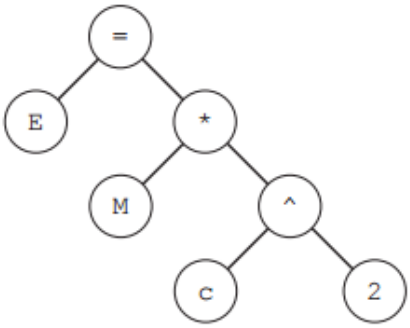
(b) What is the value of this reverse Polish expression:

`x y + p q - /`

for `x = 3, y = 9, p = 5` and `q = 1`?

Show your working. **[2]**

(c) A binary tree can be used to represent an expression or a statement.



The diagram shows the binary tree for the infix statement:

`E = M * c ^ 2`

(i) Explain how the infix form for this statement is produced using a tree traversal. **[1]**

(ii) What is the reverse Polish notation for this statement? **[1]**

(iii) Explain how the reverse Polish notation for the statement is produced using a tree traversal. **[1]**

5 (b) The following are the first few lines of a source code program written using high-level language XYZ which is about to be translated by the language compiler.

```
// program written 12 June 2013
Declare IsFound : Boolean;
Declare NoOfChildren : Integer;
Declare Count : Integer;
Constant TaxRate = 15;

// start of main program
For Count = 1 To 50

...
...
...
```

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

(i) Describe what information is contained in these tables. **[3]**

(ii) Explain how the table contents are used to translate the source code. **[2]**

Oct/Nov 2013.P32

3.4.3 Translation Software

1 (a) Convert the following infix expressions into reverse Polish notation:

(i) `(x - y) / 5` **[1]**

(ii) `2 / (4 * a + 1)` **[2]**

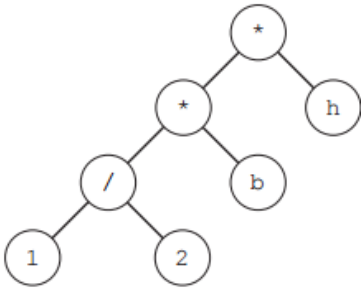
(b) What is the value of this reverse Polish expression:

`a b + c d - /`

for `a = 7, b = 5, c = 8` and `d = 2` ?

Show your working. **[2]**

(c) A binary tree can be used to represent an expression or a statement.



The diagram shows the binary tree for the infix expression:

`1 / 2 * b * h`

(i) Explain how the infix form for this expression is produced using a tree traversal. **[1]**

(ii) What is the reverse Polish notation for this expression? **[1]**

(iii) Explain how the reverse Polish notation is produced using a tree traversal. **[1]**

Oct/Nov 2013.P32

1 (a) Convert the following infix expressions into reverse Polish notation:

(i) `(p + q) / 2` **[1]**

(ii) `6 / (3 + 5 * p)` **[2]**

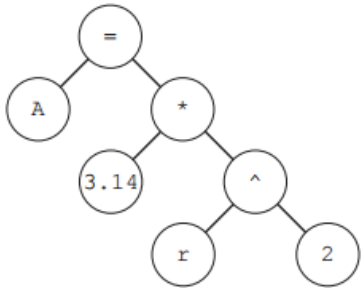
(b) What is the value of this reverse Polish expression:

`p q - r s - /`

for `p = 8, q = 2, r = 5` and `s = 3` ?

Show your working. **[2]**

(c) A binary tree can be used to represent an expression or a statement.



The diagram shows the binary tree for the infix statement:

```
A = 3.14 * r ^ 2
```

(i) Explain how the infix form for this statement is produced using a tree traversal. **[1]**

(ii) What is the reverse Polish notation for this statement? **[1]**

(iii) Explain how the reverse Polish notation is produced using a tree traversal. **[1]**

May/June 2014.P31/P32

1 The syntax of a high-level programming language is defined using Backus-Naur Form (BNF).
Five of the rules are shown below.

```
1.    <LeftBr> ::= [
2.    <RightBr> ::= ]
3.    <Digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
4.    <Integer> ::= <Digit> | <Digit><Integer> | <Integer><Digit>
5.    <ArraySubscript> ::= <LeftBr><Integer><RightBr>
```

(a) A rule may be recursive.
Explain what is meant by this and identify the rule above which is recursive. [2]

(b) (i) Use the above rules to explain why [8] is a valid . [3]
(ii) Use the above rules to explain why [3917 is an invalid . [1]

(c) The given rules permit an array subscript to start with a zero, i.e. [037] is valid.
The programming language however, does not allow an array subscript to start with a zero digit. The rules therefore need to be amended.
Complete the revised list of rules.

```
<LeftBr> ::= [
<RightBr> ::= ]
```

[4]

3.4.3 Translation Software

Oct/Nov 2014.P31/P33

1 (a) Convert the following infix form expressions into reverse Polish notation.

(i) `(a + b) /` **[1]**
(ii) `3 * (x * y + 3)` **[2]**

(b) Convert the following reverse Polish notation expressions into infix form.

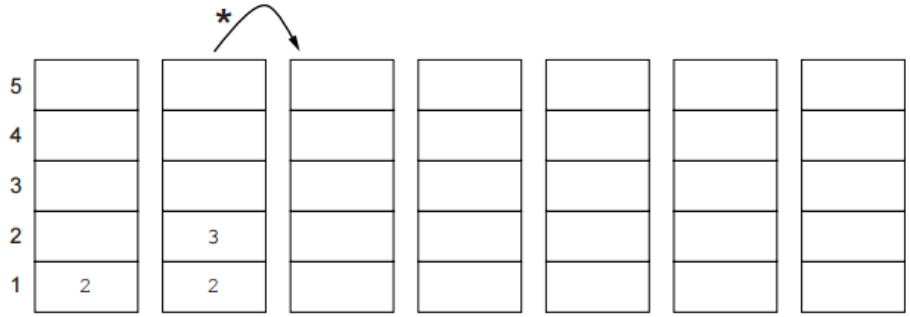
(i) `3 x y + z + *` **[1]**
(ii) `7 y ^ 6 + 2 /`
Note: the caret (^) symbol represents "to the power of". **[2]**

(c) An expression in reverse Polish notation can be evaluated on a computer system using a stack.

(i) Describe the operation of a stack. **[1]**
(ii) The expression in reverse Polish notation

`2 3 * 8 + 7 /`

is to be evaluated using a stack. The first available location on the stack is 1.
The diagram will show the changing contents of the stack as this expression is evaluated.
Complete the diagram.



[4]

Oct/Nov 2014.P32

1 (a) Convert the following infix form expressions into reverse Polish notation.

(i) `(x - y) / 4` [1]
(ii) 3 * (2 + x / 7) [2]

(b) Convert the following reverse Polish notation expressions into infix form.

(i) `4 a b + c + d + e + *` [1]
(ii) `y 2 ^ z 3 ^ + 5 /`
Note: the caret (^) symbol represents "to the power of". **[2]**

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

3.4.3 Translation Software

```
// invoicing program
// program written 21 Oct 2014
DECLARE  i : INTEGER;
DECLARE Customer(40) : STRING;
DECLARE Address: STRING;
CONSTANT DiscountRate = 5;

// start of main program
CALL InitialiseCustomerData
REPEAT
…
…
…
```

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

(i) Describe what information is contained in the keyword table. **[2]**

(ii) List three entries which must be in the keyword table for this program. **[1]**

(iii) Describe what information is contained in the symbol table. **[2]**

(iv) List three entries which will be entered in the symbol table for this program. **[1]**

(v) Explain what happens during the lexical analysis stage of compilation. Include how the contents of the keyword table and symbol table are used. **[5]**

(b) The final stage of compilation is code optimisation.

(i) Explain what is meant by code optimisation. **[2]**

(ii) Consider three assembly language instructions that were given in Question 4.

Instruction		Explanation
Op Code	Operand	
LDD	<address>	Direct addressing. Load the contents of the given address to ACC
STO	<address>	Copy the contents of ACC to the given address
INC	<register>	Add 1 to the contents of the register (ACC or IX)

Study the assembly language code below.

200	LDD 151
201	INC ACC
202	STO 151
203	LDD 151
204	INC ACC
205	STO 151

One instruction is not needed and could be removed during optimisation.
State the address of this instruction. **[1]**

May/June 2015.P31/P32

3.4.3 Translation Software

2 A set of Backus-Naur Form (BNF) rules is given as follows:

Rule number	
1	<char> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
2	<string> ::= <char> | <char><string>
3	<start> ::= 0
4	<stop> ::= 1
5	<packet> ::= <start><string><stop>

(a) State the meaning of the vertical line character (|) shown in rules 1 and 2. **[1]**

(b) A sequence of characters (a packet) is transmitted from a remote data logger to a computer.
The packet is made up of:

- one character to denote the start of the transmission
- a string of characters
- one character to denote the end of the transmission

Each packet follows the given BNF rules.
A BNF rule may be recursive.
Identify the rule above which is recursive.
Explain what is meant by recursive. **[2]**

(c) Circle whether or not each of the following sequence of characters is a valid packet.
Show how you arrived at your answer by listing, in order, the rules used.

(i) `0A1` Valid / Invalid (circle) **[2]**

(ii) `P1` Valid / Invalid (circle) **[2]**

(iii) `0TAN1` Valid / Invalid (circle) **[2]**

(iv) The rules need to change as follows.
A string with a sequence of one or more hash (#) characters is to be allowed as a valid string:
0#1 and 0####1 would both be a valid packet.
Implement this by:

- making no change to rule 1
- making the appropriate changes (if any) to rules 2, 3, 4 and 5
- adding one or more new rule(s) Show the complete set of rules below.

Rule number	
1	<char> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
2	<string> ::= .....
3	<start> ::= .....
4	<stop> ::= .....
5	<packet> ::= .....
6	.....
7	.....

[3]

May/June 2015.P33

3.4.3 Translation Software

2 (a) Backus-Naur Form (BNF) is used to define programming language syntax.
State one other method used to define programming language syntax. **[1]**

(b) A firm sells a variety of products. Each product type has a single character code:

- W – White goods, such as a washing machine
- C – Computing
- B – Books
- H – Household

All products are stocked at one of two warehouses which are coded:

- N – New Delhi
- M – Mumbai

The following BNF rules define all possible product codes.

Rule number	
1	<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
2	<producttype> ::= B | C | H | W
3	<location> ::= M | N
4	<digitstring> ::= <digit><digit><digit>
5	<productcode> ::= <producttype><digitstring><location>

(i) A BNF rule may be recursive. What is meant by a recursive rule? **[1]**

(ii) For each statement below, state whether it is TRUE or FALSE.

Statement	TRUE or FALSE
None of the given rules are recursive	
Rule 4 is recursive	
Rule 5 is recursive	

[1]

(c) Circle whether or not each of the following sequences of characters is a valid product code.
Show how you arrived at your answer by listing, in order, the rules used.

(i) `D175N` Valid / Invalid (circle). **[2]**

(ii) `W058M` Valid / Invalid (circle) **[2]**

(iii) `C86N` Valid / Invalid (circle) **[2]**

3.4.3 Translation Software

Computer Science (9608)

Oct/Nov 2015.P31/P33

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.

3.4.3 Translation Software

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. **[2]**

Oct/Nov 2015.P32

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.

3.4.3 Translation Software

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
```

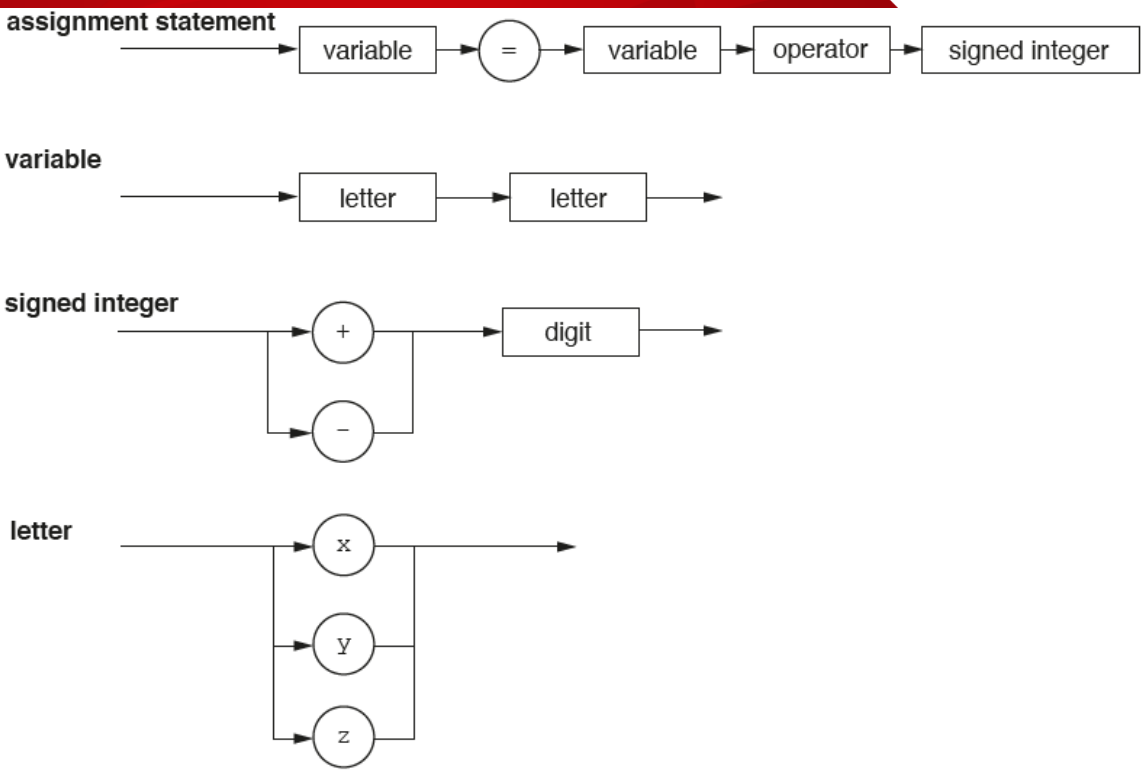
(iv) Rewrite the equivalent code, given above, following optimisation. **[3]**

May/June 2018.P31/P33

5 The following syntax diagrams show the syntax of:

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

3.4.3 Translation Software



(a) The following assignment statements are invalid.
Give the reason in each case.

(i) `xy = xy ^ c4` **[1]**

(ii) `zy = zy \ 10` **[1]**

(iii) `yy := xz ^ - 6` **[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]**