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.

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]

Computer Science 9608
Topical Past Papers
Zak ZAFAR ALI KHAN
3.4.3 Translation Software

(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. Syntax Checking / parsing **[1]**

(ii) State **two** tasks that occur at this stage. Irregular assignments. **[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. Faster execution of the code. **[1]**

(ii) What could a compiler do to optimise the following expression? [1]

```
A ← B + 2 * 6
```

(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]

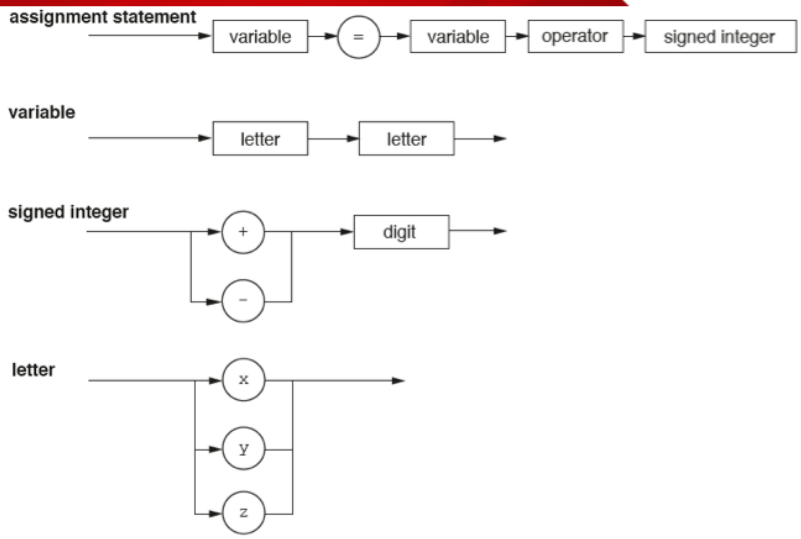
May/June 2018.P31/P33

5 The following syntax diagrams show the syntax of:

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

29-11-20

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> ::=
```