

Week 45

A-Level Computer Science

Homework bundle for this week

本周作业合集

exercises.lol

Contents 目录

Topic 16 quiz	2
Exercise sheet 16.1	5
Past-paper questions 16.1	10
Exercise sheet 16.2	15
Past-paper questions 16.2	20

A-Level Computer Science

Name: _____ Score: _____

1. Multi-tasking lets several programs appear to run at once by:
 - ☐ switching the CPU quickly between processes
 - ☐ using several monitors
 - ☐ deleting idle programs
 - ☐ compressing the programs
2. A benefit of virtual memory is that it:
 - ☐ lets the total memory used exceed the physical RAM, and isolates processes
 - ☐ makes the CPU run faster
 - ☐ removes the need for a disk
 - ☐ encrypts all data
3. An interpreter:
 - ☐ translates and runs the program one statement at a time, with no executable produced
 - ☐ always produces a stand-alone executable
 - ☐ reports all errors before running
 - ☐ is faster at run time than compiled code
4. In BNF, a terminal symbol is:
 - ☐ literal text that appears in the program
 - ☐ the name of another rule
 - ☐ always recursive
 - ☐ a syntax error
5. Spooling helps the system because:
 - ☐ print jobs queue on disk, so the CPU never waits for the slow printer
 - ☐ it encrypts print jobs
 - ☐ it speeds up the CPU clock
 - ☐ it deletes old documents
6. Round-robin scheduling gives each ready process a fixed time slice, then moves it to the back of the queue — making it fair and responsive.
 - ☐ True ☐ False

7. Evaluate the RPN expression '3 4 2 * +'.

8. Segmentation divides memory into variable-sized logical units (code, stack, heap), each with its own permissions, whereas paging uses fixed-size pages.

☐ True ☐ False

9. Reverse Polish Notation needs no brackets or precedence rules, and can be evaluated directly with a stack.

☐ True ☐ False

10. In paging, memory is divided into:

- ☐ fixed-size pages (logical) mapped to same-sized frames (physical)
- ☐ variable-sized logical segments
- ☐ single bytes
- ☐ one big block per process

1. switching the CPU quickly between processes

The OS rapidly switches the single CPU between processes so they all seem to progress together.

2. lets the total memory used exceed the physical RAM, and isolates processes

Virtual memory gives each process a private space and lets the working set exceed installed RAM by using disk.

3. translates and runs the program one statement at a time, with no executable produced

It works line by line, reporting errors as it reaches them; nothing is saved as an executable, and it is generally slower.

4. literal text that appears in the program

Terminals are literal tokens; non-terminals are names of other production rules.

5. print jobs queue on disk, so the CPU never waits for the slow printer

Spooling buffers print jobs to disk so the fast CPU is not held up by the slow printer.

6. True

Equal time slices in turn stop any one process hogging the CPU, so interactive programs stay responsive.

7. 11

Push 3, 4, 2; '*' pops 4 and 2 $\rightarrow$ 8; '+' pops 3 and 8 $\rightarrow$ 11.

8. True

Segments match the program's logical structure; pages are uniform fixed-size blocks — some systems combine both.

9. True

Push operands; each operator pops its operands and pushes the result — which is exactly how a stack machine runs.

10. fixed-size pages (logical) mapped to same-sized frames (physical)

Paging uses fixed-size pages and frames, linked by a page table. (Variable sizes are segmentation.)

16.1 Purposes of an Operating System (OS)

Name: _____ Class: _____ Date: _____

Total: 17 marks

KEY WORDS process 进程 • segmentation 分段 • thrashing 抖动 • kernel 内核 • multi-tasking 多任务

Objective

Build the skills to answer exam questions on the **operating system** 操作系统 — resource and process management, and virtual memory.

You must be able to:

- explain how an OS **maximises use of resources**
- describe **process management** (states, scheduling, context switch)
- explain **virtual memory, paging and segmentation**

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ Managing resources

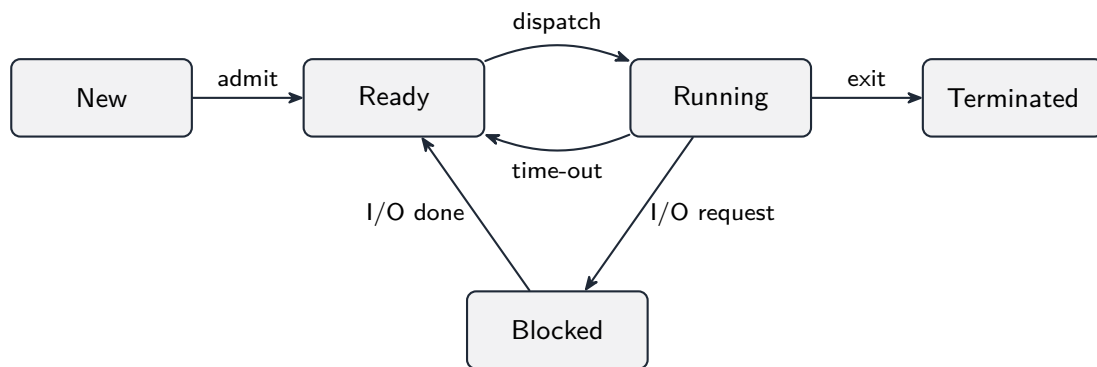
The OS shares the CPU, memory and devices fairly: **multi-tasking** (switch the CPU between processes), **memory management**, **spooling**/buffering (print jobs queue on disk), and **caching**.

■ Process management

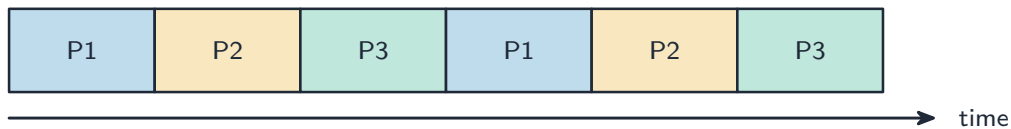
A **process** is a program in execution. The **scheduler** picks which ready process runs next. Common routines:

- **First Come First Served (FCFS)** — run in arrival order (simple, but a long job delays the rest).
- **Shortest Job First (SJF)** — run the job with the shortest expected time next (low average wait).
- **Shortest Remaining Time (SRT)** — pre-emptive SJF: switch to a newly-arrived shorter job.
- **Round robin** — each process gets a fixed **time slice** in turn, then goes to the back of the queue (fair, responsive).

A process moves between states:



each process gets a fixed time slice, then the next one runs



Round-robin scheduling shares the processor fairly

Switching from one process to another saves/restores state via the **process control block** (PCB) — a **context switch**. An **interrupt** makes the CPU pause the current process; the OS **kernel** runs the matching interrupt service routine (for an I/O-complete signal, a timer, or an error), then resumes — this is how the OS responds to events.

■ Virtual memory

Each process gets its own **virtual address space**, mapped to physical RAM. In **paging**, virtual memory is split into fixed **pages**, RAM into **frames**, linked by a **page table**. A **page fault** (page not in RAM) loads it from the **swap file**, evicting another page. This lets total memory **exceed RAM**; too many faults cause **thrashing**. **Segmentation** uses variable-sized logical segments (code, stack, heap).

2 Practice

Now apply the methods above.

2.1 State **two** resources that an operating system manages. [1]

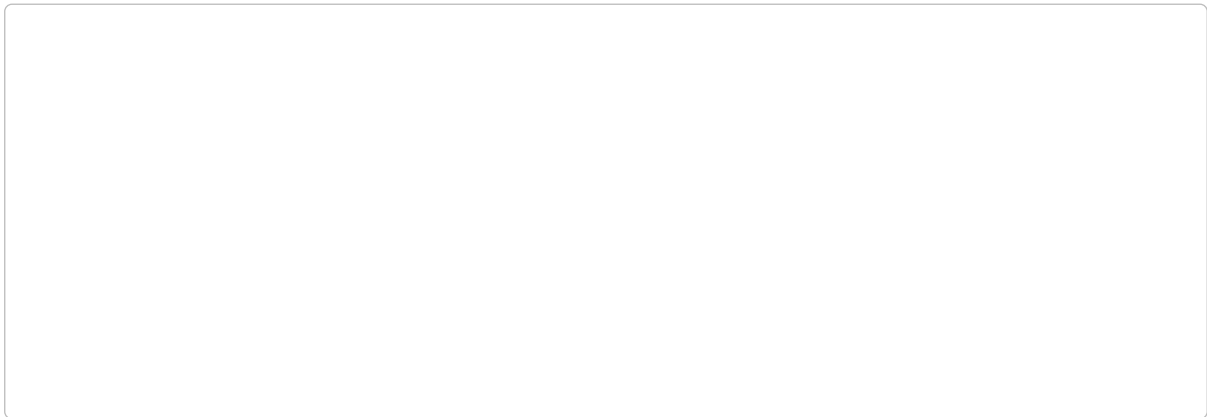
2.2 State what **round-robin** scheduling does. [1]

2.3 Name the **five** process states. [2]

2.4 State what a **context switch** is. [1]

3 Exam-style questions

3.1 Draw the **process state diagram** with the states New, Ready, Running, Blocked and Terminated, and label the transitions. [4]



3.2 Explain what happens to a **running** process when it requests **input/output**. [2]

3.3 Explain how **virtual memory** lets programs use more memory than the physical RAM. [3]

3.4 State what causes **thrashing**.

[1]

3.5 Describe the **Shortest Job First (SJF)** scheduling routine and give **one** benefit of it.

[2]

Solutions

2.1 Any two of: CPU (time), memory/RAM, disk/storage, I/O devices.

2.2 Gives each process a **fixed time slice** of CPU in turn, then moves it to the **back of the queue**.

2.3 New, Ready, Running, Blocked, Terminated.

2.4 Saving the state of one process (to its PCB) and **restoring** another's, so the CPU can switch between them.

3.1 Correct states and transitions: New→Ready (admit); Ready⇌Running (dispatch / time-out); Running→Blocked (I/O request) and Blocked→Ready (I/O done); Running→Terminated (exit).

3.2 The process moves from **Running** to **Blocked**; the scheduler then **dispatches another ready process** to the CPU while it waits.

3.3 Each process has a **virtual address space** split into **pages**; only the pages in use are kept in RAM **frames**, the rest stay in the **swap file** on disk; on a **page fault** the OS swaps a page in (evicting one if needed), so the program can be larger than physical RAM.

3.4 Too little RAM for the active pages, so the OS spends most of its time **swapping pages in and out** instead of doing useful work.

3.5 SJF runs the ready process with the **shortest expected run time** next; benefit: the **lowest average waiting time** — short jobs are not stuck behind long ones.

4 (a) A scheduling routine determines how processes are managed by the operating system.

Identify **two** scheduling routines.

1

2 [2]

(b) Describe **two** ways in which the complexities of the computer hardware are hidden from the user.

1

.....

.....

.....

2

.....

.....

..... [4]

5 The management and scheduling of processes are tasks carried out by an operating system.

(a) Identify **three** process states.

- 1
- 2
- 3 [3]

(b) Describe the function of the shortest job first scheduling routine **and** give a benefit of this routine.

Function

.....

.....

.....

.....

.....

.....

Benefit

..... [4]

6 The management and scheduling of processes are tasks carried out by an operating system.

(a) Describe **one** reason why scheduling is necessary in process management.

[2]

(b) Explain the function of the round robin scheduling routine **and** give a benefit of this routine.

Function

Benefit

[4]

9 (a) The kernel is the central component of an Operating System (OS).

Outline how the kernel of an OS acts as an interrupt handler.

[2]

(b) (i) State what is meant by the term **multi-tasking** in an Operating System.

[1]

(ii) Describe how multi-tasking is implemented in an Operating System.

[2]

8 (a) Describe the process of **segmentation** for memory management.

[4]

(b) Explain what is meant by **disk thrashing**.

..... [3]

16.2 Translation Software

Name: _____ Class: _____ Date: _____

Total: 17 marks

KEY WORDS syntax diagram 语法图 • reverse polish notation 逆波兰表示法 • lexical analysis 词法分析
• interpreter 解释器 • machine code 机器码

Objective

Build the skills to answer exam questions on **translation software** — interpreters, the stages of compilation, BNF grammar and Reverse Polish Notation.

You must be able to:

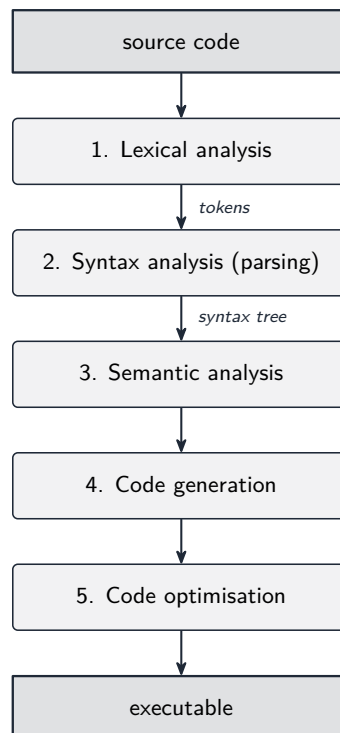
- describe how an **interpreter** runs a program and the **stages of compilation**
- read and write **BNF** production rules
- convert and evaluate **Reverse Polish Notation** (RPN)

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ Interpreter vs compiler

An **interpreter** translates and **runs each statement at once** — no executable is produced; errors stop it immediately; fast to develop, slower to run. A **compiler** translates the **whole** program to machine code first, in phases:



Each stage's job: **lexical** — group characters into **tokens** (and build the symbol table); **syntax** — check the tokens follow the grammar and build a **syntax tree**; **semantic** — check meaning (e.g. types match); **code generation** — produce the machine code; **optimisation** — make the code **faster** / **smaller** without changing what it does.

■ BNF grammar

A **BNF** rule is `<symbol> ::= alternative1 | alternative2`. Terminals are literal text; non-terminals are other rules:

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

A **syntax diagram** shows the same rules **graphically** — any path through the diagram is a valid item. You can read a definition off either form, and check whether a given value (e.g. 42, 4A) is valid against it.

■ Reverse Polish Notation

In **RPN (postfix)** the operator follows its operands — no brackets needed: infix $(3 + 4) * 2$ becomes `3 4 + 2 *`. **Evaluate** with a stack: push operands; on an operator pop two, apply, push result.

Token	Stack
3	3
4	3, 4
2	3, 4, 2
*	3, 8
+	11

2 Practice

Now apply the methods above.

2.1 State **one** difference between a **compiler** and an **interpreter**. [1]

2.2 Name the **first three** stages of compilation, in order. [2]

2.3 State what **lexical analysis** produces. [1]

2.4 Convert the infix expression $3 + 4$ to **RPN**. [1]

3 Exam-style questions

3.1 Describe the process of executing a program using an **interpreter**. [4]

3.2 Write a **BNF** rule for `<digit>` that defines a single decimal digit (0–9). [2]

3.3 Convert the infix expression $(3 + 4) * 2$ to **RPN**. [2]

3.4 Evaluate the RPN expression $5\ 2\ -\ 3\ *$ using a **stack**. Show the stack at each step. [3]

3.5 State the purpose of the **optimisation** stage of compilation. [1]

Solutions

2.1 A **compiler** translates the **whole** program to machine code first (producing an executable); an **interpreter** translates and **runs each statement** as it goes (no executable).

2.2 Lexical analysis → Syntax analysis (parsing) → Semantic analysis.

2.3 **Tokens** (keywords, identifiers, operators, literals).

2.4 3 4 +.

3.1 Any four: it reads a statement, does **lexical** then **syntax** analysis; checks it / its types; **executes** the statement's action immediately; reports any error at once and usually stops; repeats for each statement — no executable is produced — max 4.

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

3.3 3 4 + 2 *.

3.4 Push 5, push 2; - → pop 2 and 5, 5 - 2 = 3, push 3; push 3 (stack 3, 3); * → 3 * 3 = 9, result **9**.

3.5 To make the object (machine) code run **faster** and/or use **less memory**, without changing the result it produces.

8 (a) Outline the purpose of lexical analysis during the compilation of a program.

[2]

(b) Write the Reverse Polish Notation (RPN) for the given infix expression:

$(2 - 6) * (13 + 7) / 5$

[2]

(c) The RPN expression:

$d \ a \ b \ + \ * \ c \ a \ - \ /$

is to be evaluated, where:

$a = 6, \ b = 12, \ c = 15 \text{ and } d = 5.$

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

[4]

8 (a) State the purpose of the optimisation stage in the compilation of a program.

[1]

(b) Convert this Reverse Polish Notation (RPN) back to its original infix form:

a b - c + c a - * d /

[3]

(c) The RPN expression:

c a / b d - * b +

is to be evaluated, where:

a = 3, b = 16, c = 9 and d = 6.

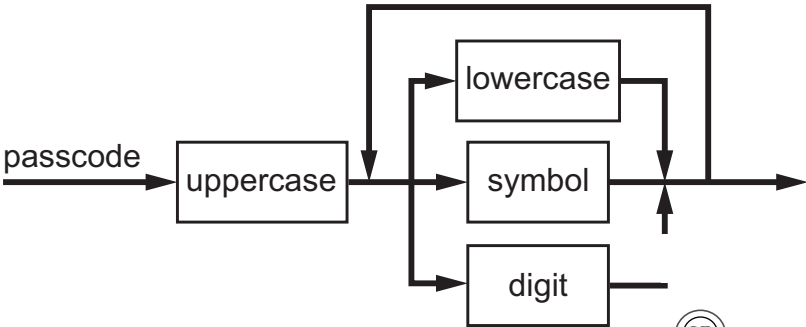
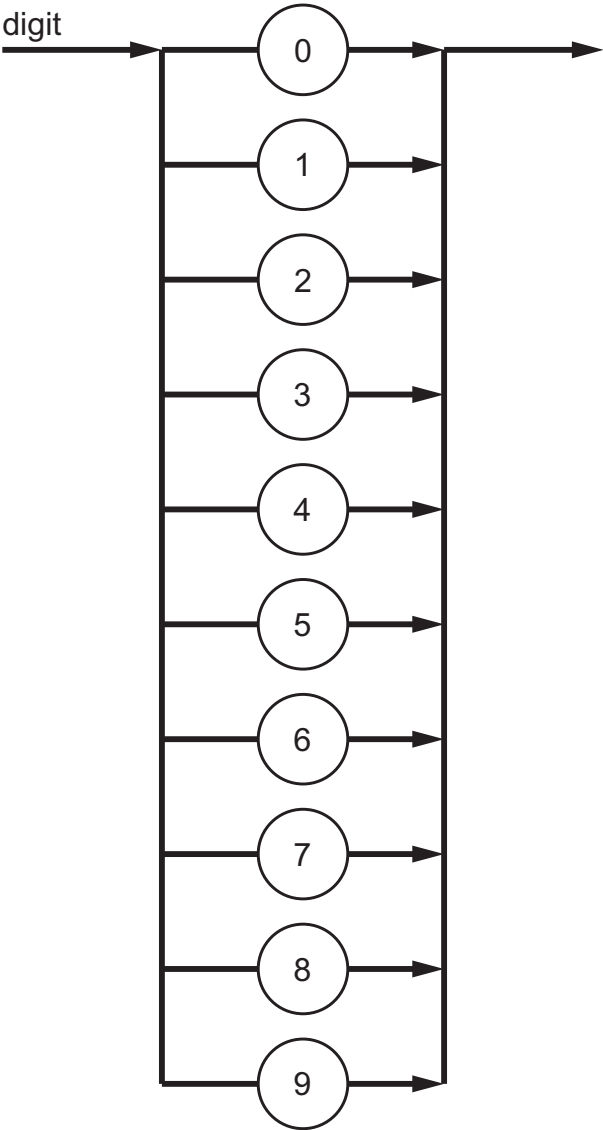
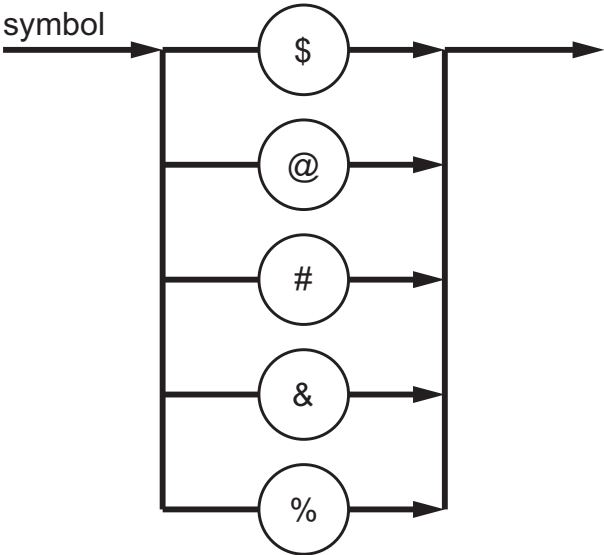
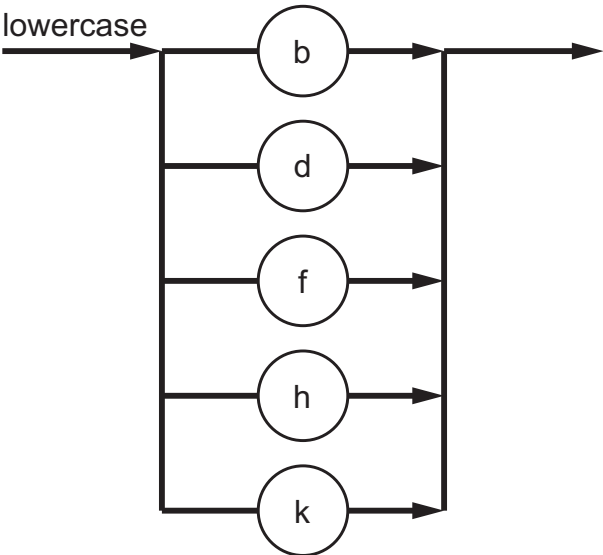
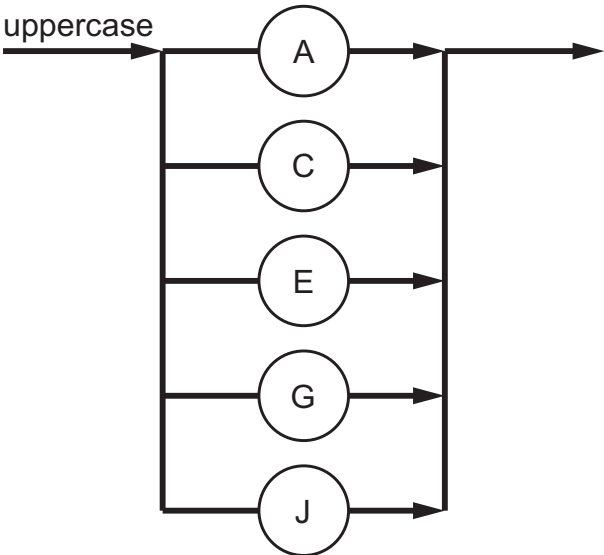
Show the changing contents of the stack as the RPN expression is evaluated.

[4]

6 Describe the process of executing a program using an interpreter.

[4]

7 Several syntax diagrams are shown.



(a) State why each **passcode** is invalid for the given syntax diagrams.

#Jd7

Reason
.....

C%6A

Reason
.....

[2]

(b) Complete the Backus-Naur Form (BNF) for <uppercase> and <passcode>.

<uppercase> ::=
.....

<passcode> ::=
.....
.....
.....
.....
.....
.....

[4]

8 Explain what is meant by **lexical analysis** during program compilation.

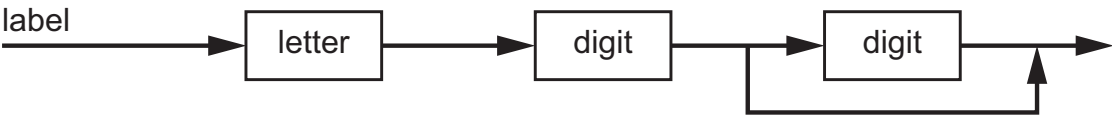
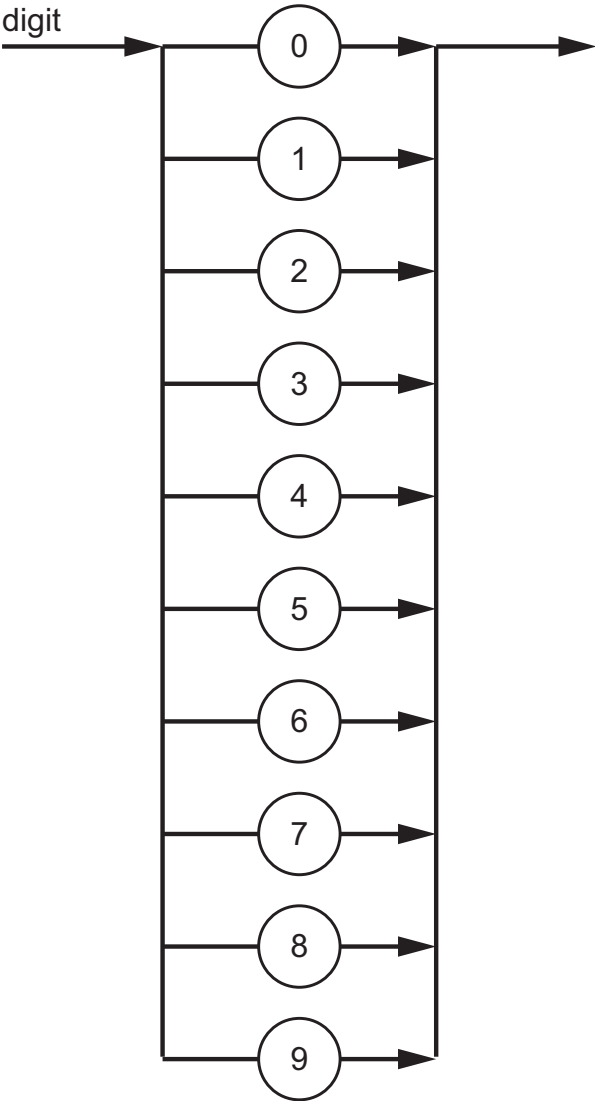
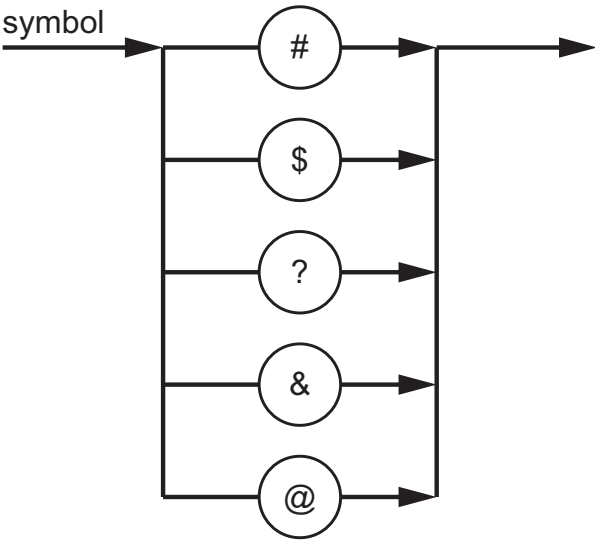
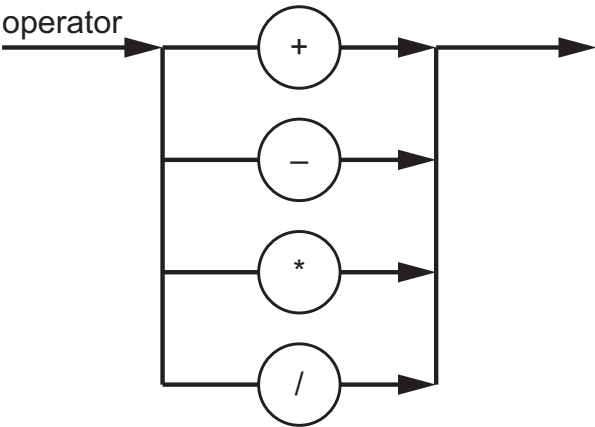
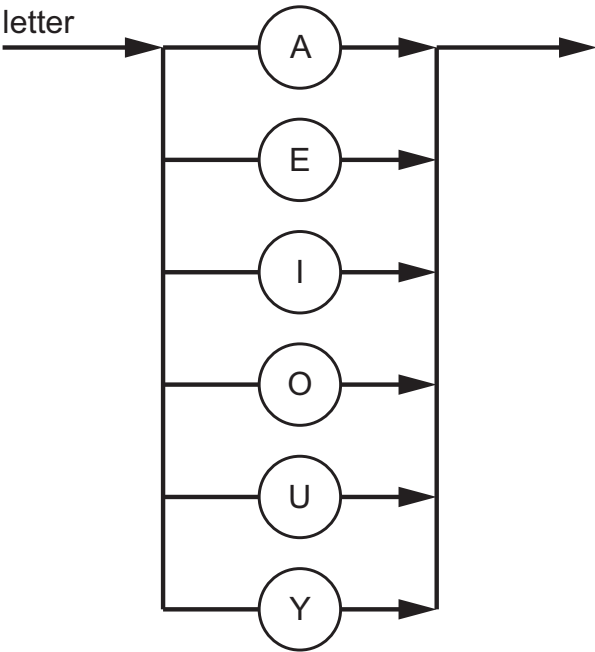
.....
.....
.....
.....
.....
.....
.....
.....

[4]

8 Explain the process of **syntax analysis** during program compilation.

[4]

10 Several syntax diagrams are shown.



(a) Complete the Backus-Naur Form (BNF) for the given syntax diagrams.

<operator> ::=

.....

<label> ::=

.....

<equation> ::=

.....

[4]

(b) A new syntax rule, **password**, is required. It must begin with a letter or a symbol, followed by a digit and end with one or two symbols.

(i) Draw a syntax diagram for **password**.

[3]

(ii) Write the BNF for **password**.

.....

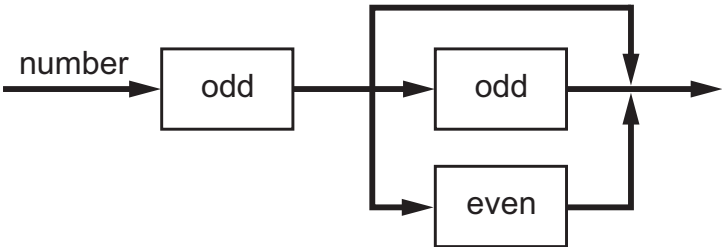
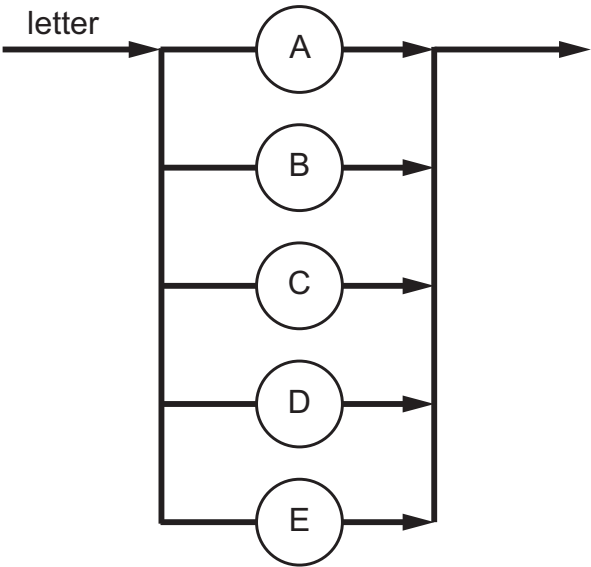
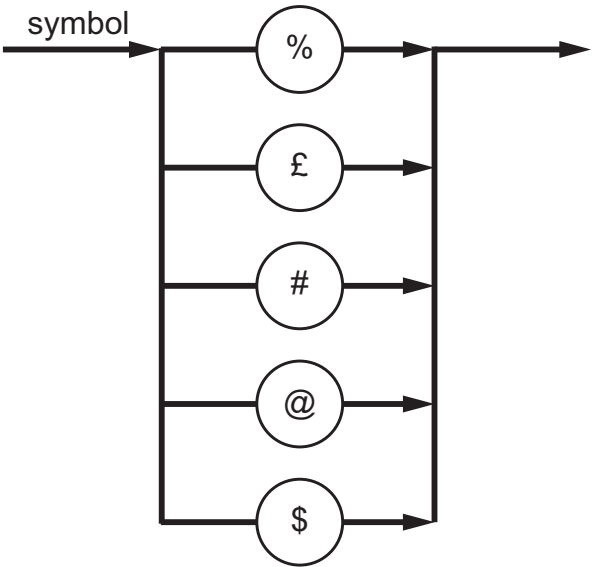
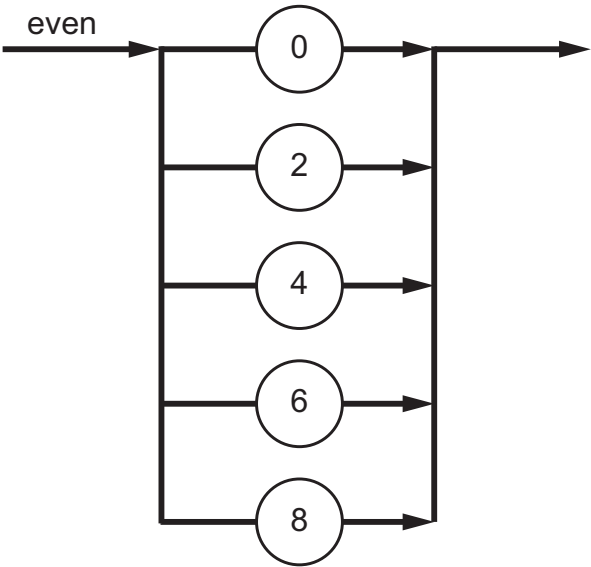
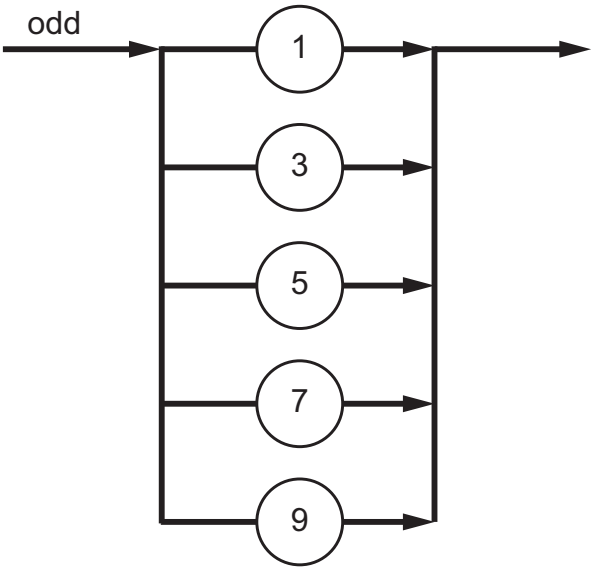
.....

.....

.....

[2]

7 Several syntax diagrams are shown.



(a) State why each number is invalid for the given syntax diagrams.

21

Reason
.....

123

Reason
.....

[2]

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

<symbol> ::=
.....

<number> ::=
.....

[2]

(c) A new syntax rule, **code**, is required. It must begin with a letter, followed by one or two numbers, and end with a symbol.

(i) Draw a syntax diagram for **code**.

[3]

(ii) Write the BNF for **code**.

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