CAMBRIDGE
International Examinations

Cambridge
International
AS & A Level

SYLLABUS

Cambridge International AS and A Level
Computer Science

9608

For examination in June and November 2016

Cambridge Advanced

Changes to syllabus for 2016

This syllabus has been updated. Significant changes to the syllabus are indicated by black vertical lines either side of the text.

Contents

1. Introduction

1.1 Why choose Cambridge?

Recognition

Cambridge International Examinations is the world's largest provider of international education programmes and qualifications for learners aged 5 to 19. We are part of Cambridge Assessment, a department of the University of Cambridge, trusted for excellence in education. Our qualifications are recognised by the world's universities and employers.

Cambridge International AS and A Levels are recognised around the world by schools, universities and employers. The qualifications are accepted as proof of academic ability for entry to universities worldwide, though some courses do require specific subjects.

Cambridge International A Levels typically take two years to complete and offer a flexible course of study that gives learners the freedom to select subjects that are right for them.

Cambridge International AS Levels often represent the first half of an A Level course but may also be taken as a freestanding qualification. The content and difficulty of a Cambridge International AS Level examination is equivalent to the first half of a corresponding Cambridge International A Level. Cambridge AS Levels are accepted in all UK universities and carry half the weighting of an A Level. University course credit and advanced standing is often available for Cambridge International AS and A Levels in countries such as the USA and Canada.

Learn more at **www.cie.org.uk/recognition**

Excellence in education

Our mission is to deliver world-class international education through the provision of high-quality curricula, assessment and services.

More than 9000 schools are part of our Cambridge learning community. We support teachers in over 160 countries who offer their learners an international education based on our curricula and leading to our qualifications. Every year, thousands of learners use Cambridge qualifications to gain places at universities around the world.

Our syllabuses are reviewed and updated regularly so that they reflect the latest thinking of international experts and practitioners and take account of the different national contexts in which they are taught.

Cambridge programmes and qualifications are designed to support learners in becoming:

- **confident** in working with information and ideas – their own and those of others
- **responsible** for themselves, responsive to and respectful of others
- **reflective** as learners, developing their ability to learn
- **innovative** and equipped for new and future challenges
- **engaged** intellectually and socially, ready to make a difference.

Support for teachers

A wide range of support materials and resources is available for teachers and learners in Cambridge schools. Resources suit a variety of teaching methods in different international contexts. Through subject discussion forums and training, teachers can access the expert advice they need for teaching our qualifications. More details can be found in Section 2 of this syllabus and at **www.cie.org.uk/teachers**

Support for exams officers

Exams officers can trust in reliable, efficient administration of exam entries and excellent personal support from our customer services. Learn more at **www.cie.org.uk/examsofficers**

Not-for-profit, part of the University of Cambridge

We are a not-for-profit organisation where the needs of the teachers and learners are at the core of what we do. We continually invest in educational research and respond to feedback from our customers in order to improve our qualifications, products and services.

Our systems for managing the provision of international qualifications and education programmes for learners aged 5 to 19 are certified as meeting the internationally recognised standard for quality management, ISO 9001:2008. Learn more at **www.cie.org.uk/ISO9001**

1.2 Why choose Cambridge International AS and A Level?

Cambridge International AS and A Levels are international in outlook, but retain a local relevance. The syllabuses provide opportunities for contextualised learning and the content has been created to suit a wide variety of schools, avoid cultural bias and develop essential lifelong skills, including creative thinking and problem-solving.

Our aim is to balance knowledge, understanding and skills in our programmes and qualifications to enable candidates to become effective learners and to provide a solid foundation for their continuing educational journey. Cambridge International AS and A Levels give learners building blocks for an individualised curriculum that develops their knowledge, understanding and skills.

Schools can offer almost any combination of 60 subjects, and learners can specialise or study a range of subjects, ensuring a breadth of knowledge. Giving learners the power to choose helps motivate them throughout their studies.

Through our professional development courses and our support materials for Cambridge International AS and A Levels, we provide the tools to enable teachers to prepare learners to the best of their ability and work with us in the pursuit of excellence in education.

Cambridge International AS and A Levels have a proven reputation for preparing learners well for university, employment and life. They help develop the in-depth subject knowledge and understanding which are so important to universities and employers.

Learners studying Cambridge International AS and A Levels have the opportunities to:

- acquire an in-depth subject knowledge
- develop independent thinking skills
- apply knowledge and understanding to new as well as familiar situations
- handle and evaluate different types of information sources
- think logically and present ordered and coherent arguments
- make judgements, recommendations and decisions
- present reasoned explanations, understand implications and communicate them clearly and logically
- work and communicate in English.

Guided learning hours

Cambridge International A Level syllabuses are designed on the assumption that learners have about 360 guided learning hours per subject over the duration of the course. Cambridge International AS Level syllabuses are designed on the assumption that learners have about 180 guided learning hours per subject over the duration of the course. This is for guidance only and the number of hours required to gain the qualification may vary according to local curricular practice and the learners' prior experience of the subject.

1.3 Why choose Cambridge International AS and A Level Computer Science?

Cambridge International AS Level and A Level Computer Science are accepted by universities and employers as proof of essential knowledge and ability.

This syllabus is designed to give greater flexibility both to teachers and to learners. It is envisaged that learners will use the skills and knowledge of computer science acquired through this course in one of three ways:

- to provide a general understanding and perspective of the development of computer technology and systems, which will inform their decisions and support their participation in an increasingly technologically dependent society
- to provide the necessary skills and knowledge to seek employment in areas that use computer science
- to develop their knowledge and understanding of computer science through entry to higher education, where this qualification will provide a useful foundation for further study of computer science or more specialist aspects of computer science.

Prior learning

Candidates beginning this course are not expected to have studied computer science or ICT previously.

Progression

Cambridge International A Level Computer Science provides a suitable foundation for the study of computer science or related courses in higher education. Equally, it is suitable for candidates intending to pursue careers or further study in computer science or ICT, or as part of a course of general education.

Cambridge International AS Level Computer Science constitutes the first half of the Cambridge International A Level course in computer science and provides a suitable foundation for the study of computer science at Cambridge International A Level and then for related courses in higher education. Depending on local university entrance requirements, it may permit or assist progression directly to university courses in Computer Science or some other subjects. It is also suitable for candidates intending to pursue careers or further study in computer science or ICT, or as part of a course of general education.

1.4 Cambridge AICE (Advanced International Certificate of Education) Diploma

Cambridge AICE Diploma is the group award of the Cambridge International AS and A Level. It gives schools the opportunity to benefit from offering a broad and balanced curriculum by recognising the achievements of candidates who pass examinations in three different curriculum groups:

- Mathematics and Science (Group 1)
- Languages (Group 2)
- Arts and Humanities (Group 3)

A Cambridge International A Level counts as a double-credit qualification and a Cambridge International AS Level counts as a single-credit qualification within the Cambridge AICE Diploma award framework.

To be considered for an AICE Diploma, a candidate must earn the equivalent of six credits by passing a combination of examinations at either double credit or single credit, with at least one course coming from each of the three curriculum groups.

Computer Science (9608) is in Group 1, Mathematics and Science.

Credits gained from Cambridge AS Level Global Perspectives (8987) or Cambridge Pre-U Global Perspectives and Independent Research (9766) can be counted towards the Cambridge AICE Diploma, but candidates must also gain at least one credit from each of the three curriculum groups to be eligible for the award.

Learn more about the Cambridge AICE Diploma at **www.cie.org.uk/qualifications/academic/uppersec/aice**

The Cambridge AICE Diploma is awarded from examinations administered in the June and November series each year.

1.5 How can I find out more?

If you are already a Cambridge school

You can make entries for this qualification through your usual channels. If you have any questions, please contact us at **info@cie.org.uk**

If you are not yet a Cambridge school

Learn about the benefits of becoming a Cambridge school at **www.cie.org.uk/startcambridge**. Email us at **info@cie.org.uk** to find out how your organisation can register to become a Cambridge school.

2. Teacher support

2.1 Support materials

Cambridge syllabuses, past question papers and examiner reports to cover the last examination series are on the *Syllabus and Support Materials* DVD, which we send to all Cambridge schools.

You can also go to our public website at **www.cie.org.uk/alevel** to download current and future syllabuses together with specimen papers or past question papers and examiner reports from one series.

For teachers at registered Cambridge schools a range of additional support materials for specific syllabuses is available from Teacher Support, our secure online support for Cambridge teachers. Go to **http://teachers.cie.org.uk** (username and password required).

2.2 Resource lists

We work with publishers providing a range of resources for our syllabuses including textbooks, websites, CDs, etc. Any endorsed, recommended and suggested resources are listed on both our public website and on Teacher Support.

The resource lists can be filtered to show all resources or just those which are endorsed or recommended by Cambridge. Resources endorsed by Cambridge go through a detailed quality assurance process and are written to align closely with the Cambridge syllabus they support.

2.3 Training

We offer a range of support activities for teachers to ensure they have the relevant knowledge and skills to deliver our qualifications. See **www.cie.org.uk/events** for further information.

3. Syllabus content at a glance

Section	Topics
Section 1 Theory Fundamentals	1.1 Information representation 1.1.1 Number representation 1.1.2 Images 1.1.3 Sound 1.1.4 Video 1.1.5 Compression techniques 1.2 Communication and Internet technologies 1.2.1 Networks 1.2.2 IP addressing 1.2.3 Client- and server-side scripting 1.3 Hardware 1.3.1 Input, output and storage devices 1.3.2 Main memory 1.3.3 Logic gates and logic circuits 1.4 Processor fundamentals 1.4.1 CPU architecture 1.4.2 The fetch-execute cycle 1.4.3 The processor's instruction set 1.4.4 Assembly language 1.5 System software 1.5.1 Operating system 1.5.2 Utility programs 1.5.3 Library programs 1.5.4 Language translators 1.6 Security, privacy and data integrity 1.6.1 Data security 1.6.2 Data integrity 1.7 Ethics and ownership 1.7.1 Ethics 1.7.2 Ownership 1.8 Database and data modelling 1.8.1 Database Management Systems (DBMS) 1.8.2 Relational database modelling 1.8.3 Data Definition Language (DDL) and Data Manipulation Language (DML)

Section 2 Fundamental Problem-solving and Programming	2.1 Algorithm design and problem-solving 2.1.1 Algorithms 2.1.2 Structure chart 2.1.3 Corrective maintenance 2.1.4 Adaptive maintenance 2.2 Data representation 2.2.1 Data types 2.2.2 Arrays 2.2.3 Files 2.3 Programming 2.3.1 Programming basics 2.3.2 Transferable skills 2.3.3 Selection 2.3.4 Iteration 2.3.5 Built-in functions 2.3.6 Structured programming 2.4 Software development 2.4.1 Programming 2.4.2 Program testing 2.4.3 Testing strategies

Section 3 Advanced Theory	3.1 Data representation 3.1.1 User-defined data types 3.1.2 File organisation and access 3.1.3 Real numbers and normalised floating-point representation 3.2 Communication and Internet technologies 3.2.1 Protocols 3.2.2 Circuit switching, packet switching and routers 3.2.3 Local Area Networks (LAN) 3.3 Hardware 3.3.1 Logic gates and circuit design 3.3.2 Boolean algebra 3.3.3 Karnaugh Maps 3.3.4 Flip-flops 3.3.5 RISC processors 3.3.6 Parallel processing 3.4 System software 3.4.1 Purposes of an operating system (OS) 3.4.2 Virtual machine 3.4.3 Translation software 3.5 Security 3.5.1 Asymmetric keys and encryption methods 3.5.2 Digital signatures and digital certificates 3.5.3 Encryption protocols 3.5.4 Malware 3.6 Monitoring and control systems 3.6.1 Overview of monitoring and control systems 3.6.2 Bit manipulation to monitor and control devices

Section 4 Further Problem-solving and Programming Skills	4.1 Computational thinking and problem-solving 4.1.1 Abstraction 4.1.2 Algorithms 4.1.3 Abstract Data Types (ADT) 4.1.4 Recursion 4.2 Algorithm design methods 4.2.1 Decision tables 4.2.2 Jackson Structured Programming (JSP) 4.2.3 State-transition diagrams 4.3 Further programming 4.3.1 Programming paradigms Low-level programming Imperative programming Object-oriented programming Declarative programming 4.3.2 File processing 4.3.3 Exception handling 4.3.4 Use of development tools / programming environments 4.4 Software development 4.4.1 Stages of software development 4.4.2 Testing 4.4.3 Project management

4. Assessment at a glance

For Cambridge International AS and A Level Computer Science, candidates may choose:

- to take Papers 1, 2, 3 and 4 in the same examination series, leading to the full Cambridge International A Level
- to follow a **staged** assessment route by taking Papers 1 and 2 (for the AS Level qualification) in one series, then Papers 3 and 4 (for the full Cambridge International A Level) in a later series
- to take Papers 1 and 2 only (for the AS Level qualification).

Components All candidates take	Weighting (%)	
	AS	A
Paper 1 Theory Fundamentals This written paper contains short-answer and structured questions. There is no choice of questions. 75 marks Externally assessed 1 hour 30 minutes	50	25
Paper 2 Fundamental Problem-solving and Programming Skills This written paper contains short-answer and structured questions. There is no choice of questions. Topics will include those given in the pre-release material.[1] 75 marks Externally assessed 2 hours	50	25
Paper 3 Advanced Theory This written paper contains short-answer and structured questions. There is no choice of questions. 75 marks Externally assessed 1 hour 30 minutes	–	25
Paper 4 Further Problem-solving and Programming Skills This written paper contains short-answer and structured questions. There is no choice of questions. Topics will include those given in the pre-release material.[1] 75 marks Externally assessed 2 hours	–	25

Advanced Subsidiary (AS) forms 50% of the assessment weighting of the full Advanced (A) Level.

[1] The pre-release material will be made available to Centres the January before the June examination, and the July before the November examination. Candidates are not permitted to bring any prepared material into the examination.

Availability

This syllabus is examined in the June and November examination series.

This syllabus is available to private candidates.

Detailed timetables are available from **www.cie.org.uk/examsofficers**

Centres in the UK that receive government funding are advised to consult the Cambridge website **www.cie.org.uk** for the latest information before beginning to teach this syllabus.

Combining this with other syllabuses

Candidates can combine this syllabus in an examination series with any other Cambridge syllabus, except syllabuses with the same title at the same level.

5. Syllabus aims and assessment objectives

5.1 Syllabus aims

The aims of a course based on Cambridge International AS and AL Computer Science, whether leading to an AS or A Level qualification are:

- to develop computational thinking
- to develop an understanding of the main principles of solving problems using computers
- to develop an understanding that every computer system is made up of subsystems, which in turn consist of further subsystems
- to develop an understanding of the component parts of computer systems and how they interrelate, including software, data, hardware, communications and people
- to acquire the skills necessary to apply this understanding to develop computer-based solutions to problems.

Computer science is the study of the foundational principles and practices of computation and computational thinking and their application in the design and development of computer systems.

This syllabus aims to encourage the development of computational thinking, that is thinking about what can be computed and how by the use of abstraction and decomposition. It includes consideration of the data required. Learning computational thinking involves learning to program, by writing computer code, because this is the means by which computational thinking is expressed.

5.2 Assessment objectives

Cambridge International AS and A Level Computer Science has two assessment objectives:

AO1 Knowledge with understanding

- show understanding of the characteristics and methods of operation of component parts of computer systems (hardware, software, communication) and their subsystems
- describe, explain and use various different methods of representing data for use in computer systems
- comment critically on ethical issues arising from the use of computer solutions.

AO2 Skills

- apply knowledge with understanding to computational problems
- select, justify and apply appropriate techniques and principles to develop data structures and algorithms for the solutions of computational problems
- design, implement, document and evaluate an effective solution using appropriate hardware, software and programming languages.

5.3 Relationship between assessment objectives and components

The approximate weightings allocated to each of the assessment objectives are summarised below.

Assessment objective	Paper 1 (%)	Paper 2 (%)	Paper 3 (%)	Paper 4 (%)	AS Level (%)	A Level (%)
AO1 Knowledge with understanding	20	5	20	5	50	50
AO2 Skills	5	20	5	20	50	50

6. Syllabus content

This syllabus is set out in the form of teaching sections. Each teaching section is assessed by its associated paper. The AS Level syllabus consists of teaching Sections 1 and 2 only, and the A Level syllabus consists of all four teaching sections.

The subject content for each section is shown below.

Syllabus content section	Paper	Section title
1	1	Theory Fundamentals
2	2	Fundamental Problem-solving and Programming Skills
3	3	Advanced Theory
4	4	Further Problem-solving and Programming Skills

Each section is presented as a set of sub-sections, each with details of content and associated learning outcomes.

Section 1 Theory Fundamentals

1.1 Information representation

Candidates should be able to:

1.1.1 Number representation

- show understanding of the basis of different number systems and use the binary, denary and hexadecimal number system
- convert a number from one number system to another
- express a positive or negative integer in two's complement form
- show understanding of, and be able to represent, character data in its internal binary form depending on the character set used (Candidates will not be expected to memorise any particular character codes but must be familiar with ASCII and Unicode.)
- express a denary number in Binary Coded Decimal (BCD) and vice versa
- describe practical applications where BCD is used

1.1.2 Images

- show understanding of how data for a bitmapped image is encoded
- use the terminology associated with bitmaps: pixel, file header, image resolution, screen resolution
- perform calculations estimating the file size for bitmapped images of different resolutions
- show understanding of how data for a vector graphic is represented and encoded
- use the terminology associated with vector graphics: drawing object, property and drawing list
- show understanding of how typical features found in bitmapped and vector graphics software are used in practice
- justify where bitmapped graphics and/or vector graphics are appropriate for a given task

1.1.3 Sound

- show understanding of how sound is represented and encoded
- use the associated terminology: sampling, sampling rate, sampling resolution
- show understanding of how file sizes depend on sampling rate and sampling resolution
- show understanding of how typical features found in sound-editing software are used in practice

1.1.4 Video

- Show understanding of the characteristics of video streams:
  - the frame rate (frames/second)
  - interlaced and progressive encoding
  - video interframe compression algorithms and spatial and temporal redundancy
  - multimedia container formats

1.1.5 Compression techniques

- show understanding of how digital data can be compressed, using either 'lossless' (including run-length encoding – RLE) or 'lossy' techniques

1.2 Communication and Internet technologies

Candidates should be able to:

1.2.1 Networks

- explain the client-server model of networked computers
- give examples of applications which use the client-server model
- describe what is meant by the World Wide Web (WWW) and the Internet
- explain how hardware is used to support the Internet: networks, routers, gateways, servers
- explain how communication systems are used to support the Internet: The Public Service Telephone Network (PSTN), dedicated lines, cell phone network
- explain the benefits and drawbacks of using copper cable, fibre-optic cabling, radio waves, microwaves, satellites
- show understanding of bit streaming (both real-time and on-demand)
- show understanding of the importance of bit rates/broadband speed on bit streaming

1.2.2 IP addressing

- explain the format of an IP address and how an IP address is associated with a device on a network
- explain the difference between a public IP address and a private IP address and the implication for security
- explain how a Uniform Resource Locator (URL) is used to locate a resource on the World Wide Web (WWW) and the role of the Domain Name Service

1.2.3 Client- and server-side scripting

- describe the sequence of events executed by the client computer and web server when a web page consisting only of HTML tags is requested and displayed by a browser
  - Client-side
    - recognise and identify the purpose of some simple JavaScript code
    - describe the sequence of events executed by the client computer and web server when a web page with embedded client-side code is requested and displayed by a browser
    - show understanding of the typical use of client-side code in the design of an application
  - Server-side
    - recognise and identify the purpose of some simple PHP code
    - describe the sequence of events executed by the client computer and web server when a web page with embedded server-side code is requested and displayed by a browser
    - show understanding that an appropriately designed web application for accessing database data makes use of server-side scripting

1.3 Hardware

Candidates should be able to:

1.3.1 Input, output and storage devices

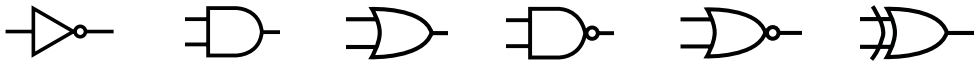
- identify hardware devices used for input, output, secondary storage
- show understanding of the basic internal operation of the following specific types of device:
  - keyboard
  - trackerball mouse
  - optical mouse
  - scanner
  - inkjet printer
  - laser printer
  - speakers
  - hard disk
  - solid state (flash) memory
  - optical discs
  - sensors
  - actuators
- show understanding of the need for secondary (including removable) storage

1.3.2 Main memory

- show understanding of the need for primary storage
  - explain the differences between RAM and ROM memory
  - explain the differences between Static RAM (SRAM) and Dynamic RAM (DRAM)

1.3.3 Logic gates and logic circuits

- use the following logic gate symbols:



NOT AND OR NAND NOR XOR

- understand and define the functions of NOT, AND, OR, NAND, NOR and XOR (EOR) gates including the binary output produced from all the possible binary inputs (all gates, except the NOT gate, will have two inputs only)
- construct the truth table for each of the logic gates above
- construct a logic circuit from either:
  - a problem statement
  - a logic expression
- construct a truth table from either:
  - a logic circuit
  - a logic expression
- show understanding that some circuits can be constructed with fewer gates to produce the same outputs

1.4 Processor fundamentals

Candidates should be able to:

1.4.1 CPU architecture

- show understanding of the basic Von Neumann model for a computer system and the stored program concept
- show understanding of the roles carried out by registers, including the difference between general purpose and special purpose registers: Program Counter, Memory Data Register, Memory Address Register, Index Register, Current Instruction Register and Status Register
- show understanding of the roles carried out by the Arithmetic and Logic Unit (ALU), Control Unit and system clock
- show understanding of how data are transferred between various components of the computer system using the address bus, data bus and control bus
- show understanding of how the bus width and clock speed are factors that contribute to the performance of the computer system
- show understanding of the need for ports, for example Universal Serial Bus (USB), to provide the connection to peripheral devices

1.4.2 The fetch-execute cycle

- describe the stages of the fetch-execute cycle
- show understanding of 'register transfer' notation
- describe how interrupts are handled

1.4.3 The processor's instruction set

- show understanding that the set of instructions are grouped into instructions for:
  - data movement (register to main memory and vice versa)
  - input and output of data
  - arithmetic operations
  - unconditional and conditional jump instructions
  - compare instructions
  - modes of addressing: immediate, direct, indirect, indexed, relative

  (No particular instruction set will be expected but candidates should be familiar with the type of instructions given in the table on page 20.)

1.4.4 Assembly language

- show understanding of the relationship between assembly language and machine code, including symbolic and absolute addressing, directives and macros
- describe the different stages of the assembly process for a 'two-pass' assembler for a given simple assembly language program
- trace a given simple assembly language program

Instruction		Explanation
Op Code	Operand	
LDM	#n	Immediate addressing. Load the number n to ACC
LDD	<address>	Direct addressing. Load the contents of the given address to ACC
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC
LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents of this calculated address to ACC
LDR	#n	Immediate addressing. Load the number n to IX
STO	<address>	Store the contents of ACC at the given address
ADD	<address>	Add the contents of the given address to the ACC
INC	<register>	Add 1 to the contents of the register (ACC or IX)
DEC	<register>	Subtract 1 from the contents of the register (ACC or IX)
JMP	<address>	Jump to the given address
CMP	<address>	Compare the contents of ACC with the contents of <address>
CMP	#n	Compare the contents of ACC with number n
JPE	<address>	Following a compare instruction, jump to <address> if the compare was True
JPN	<address>	Following a compare instruction, jump to <address> if the compare was False
IN		Key in a character and store its ASCII value in ACC
OUT		Output to the screen the character whose ASCII value is stored in ACC
END		Return control to the operating system

All questions will assume there is only one general purpose register available (Accumulator)

ACC denotes Accumulator
IX denotes Index Register

# denotes immediate addressing
B denotes a binary number, e.g. B01001010
& denotes a hexadecimal number, e.g. &4A

1.5 System software

Candidates should be able to:

1.5.1 Operating system

- describe why a computer system requires an operating system
- explain the key management tasks carried out by the operating system

1.5.2 Utility programs

- show an understanding of the need for typical utility software used by a PC computer system:
  - disk formatter
  - virus checker
  - defragmenter software
  - disk contents analysis/disk repair software
  - file compression
  - backup software

1.5.3 Library programs

- show an understanding that software under development is often constructed using existing code from program libraries
- describe the benefits to the developer of software constructed using library files, including Dynamic Link Library (DLL) files
- draw on experience of the writing of programs which include library routines

1.5.4 Language translators

- show an understanding of the need for:
  - assembler software for the translation of an assembly language program
  - a compiler for the translation of a high-level language program
  - an interpreter for execution of a high-level language program
- explain the benefits and drawbacks of using either a compiler or interpreter
- show awareness that high-level language programs may be partially compiled and partially interpreted, such as Java

1.6 Security, privacy and data integrity

Candidates should be able to:

1.6.1 Data security

- explain the difference between the terms security, privacy and integrity of data
- show appreciation of the need for both the security of data and the security of the computer system
- describe security measures designed to protect computer systems, ranging from the stand-alone PC to a network of computers, including:
  - user accounts
  - firewalls
  - general authentication techniques, including the use of passwords and digital signatures
- describe security measures designed to protect the security of data, including:
  - data backup
  - a disk-mirroring strategy
  - encryption
  - access rights to data (authorisation)
- show awareness of what kind of errors can occur and what can be done about them

1.6.2 Data integrity

- describe error detection and correction measures designed to protect the integrity of data, including:
  - data validation
  - data verification for data entry
  - data verification during data transfer, including
    - parity check
    - checksum check

1.7 Ethics and ownership

Candidates should be able to:

1.7.1 Ethics and the computing professional

- show a basic understanding of ethics
- explain how ethics may impact on the job role of the computing professional
- show understanding of the eight principles listed in the ACM/IEEE Software Engineering Code of Ethics
- demonstrate the relevance of these principles to some typical software developer workplace scenarios
- show understanding of the need for a professional code of conduct for a computer system developer

1.7.2 Ownership of software and data

- show understanding of the concept of ownership and copyright
- describe the need for legislation to protect ownership, usage and copyright
- discuss measures to restrict access to data made available through the Internet and World Wide Web
- show understanding of the implications of different types of software licensing: Free Software Foundation, the Open Source Initiative, shareware and commercial software

1.8 Database and data modelling

Candidates should be able to:

1.8.1 Database Management Systems (DBMS)

- show understanding of the limitations of using a file-based approach for the storage and retrieval of data
- describe the features of a relational database which address the limitations of a file-based approach
- show understanding of the features provided by a DBMS to address the issues of:
  - data management, including maintaining a data dictionary
  - data modelling
  - logical schema
  - data integrity
  - data security, including backup procedures and the use of access rights to individuals/groups of users
- show understanding of how software tools found within a DBMS are used in practice:
  - developer interface
  - query processor
- show awareness that high-level languages provide accessing facilities for data stored in a database

1.8.2 Relational database modelling

- show understanding of, and use, the terminology associated with a relational database model: entity, table, tuple, attribute, primary key, candidate key, foreign key, relationship, referential integrity, secondary key and indexing
- produce a relational design from a given description of a system
- use an entity-relationship diagram to document a database design
- show understanding of the normalisation process: First (1NF), Second (2NF) and Third Normal Form (3NF)
- explain why a given set of database tables are, or are not, in 3NF
- make the changes to a given set of tables which are not in 3NF to produce a solution in 3NF, and justify the changes made

1.8.3 Data Definition Language (DDL) and Data Manipulation Language (DML)

- show understanding that DBMS software carries out:
  - all creation/modification of the database structure using its DDL
  - query and maintenance of data using its DML
- show understanding that the industry standard for both DDL and DML is Structured Query Language (SQL)
  - show understanding of a given SQL script
  - write simple SQL (DDL) commands using a sub-set of commands for:
    - creating a database (CREATE DATABASE)
    - creating a table definition (CREATE TABLE)
    - changing a table definition (ALTER TABLE)
    - adding a primary key or foreign key to a table (ADD PRIMARY KEY)
  - write a SQL script for querying or modifying data (DML) which are stored in (at most two) database tables
    - Queries:
    - SELECT, FROM, WHERE, ORDER BY, GROUP BY, INNER JOIN
  - Data maintenance:
    - INSERT INTO, DELETE FROM, UPDATE

Section 2 Fundamental Problem-solving and Programming Skills

2.1 Algorithm design and problem-solving

Candidates should be able to:

2.1.1 Algorithms

- show understanding that an algorithm is a solution to a problem expressed as a sequence of defined steps
- use suitable identifier names for the representation of data used by a problem
  - summarise identifier names using an identifier table
- show understanding that many algorithms are expressed using the four basic constructs of assignment, sequence, selection and repetition
- show understanding that simple algorithms consist of input, process, output at various stages
- document a simple algorithm using:
  - structured English
  - pseudocode (on the examination paper, any given pseudocode will be presented using the Courier New font)
  - program flowchart
- derive pseudocode or a program flowchart from a structured English description of a problem
- derive pseudocode from a given program flowchart or vice versa
- use the process of stepwise refinement to express an algorithm to a level of detail from which the task may be programmed
- decompose a problem into sub-tasks leading to the concept of a program module (procedure/function)
- show an appreciation of why logic statements are used to define parts of an algorithm solution
- use logic statements to define parts of an algorithm solution

2.1.2 Structure chart

- use a structure chart to express the parameters passed between the various modules/procedures/functions which are part of the algorithm design
- describe the purpose of a structure chart
- construct a structure chart for a given problem
- derive equivalent pseudocode from a structure chart

2.1.3 Corrective maintenance

- perform white-box testing by:
  - selecting suitable data
  - using a trace table
- identify any error(s) in the algorithm by using the completed trace table
- amend the algorithm if required

2.1.4 Adaptive maintenance

- make amendments to an algorithm and data structure in response to specification changes
- analyse an existing program and make amendments to enhance functionality

2.2 Data representation

Candidates should be able to:

2.2.1 Data types

- select appropriate data types for a problem solution
- use in practical programming the data types that are common to procedural high-level languages: integer, real, char, string, Boolean, date (pseudocode will use the following data types: `INTEGER, REAL, CHAR, STRING, BOOLEAN, DATE, ARRAY, FILE`)
- show understanding of how character and string data are represented by software including the ASCII and Unicode character sets

2.2.2 Arrays

- use the technical terms associated with arrays including upper and lower bound
- select a suitable data structure (1D or 2D array) to use for a given task
- use pseudocode for 1D and 2D arrays (pseudocode will use square brackets to contain the array subscript, for example a 1D array as `A[1:n]` and a 2D array as `C[1:m, 1:n]`)
- write program code using 1D and 2D arrays
- write algorithms/program code to process array data including:
  - sorting using a bubble sort
  - searching using a linear search
- given pseudocode will use the following structures:

```
DECLARE <identifier> : ARRAY[<lbound>:<ubound>] OF <datatype>
DECLARE <identifier> : ARRAY[<lbound1>:<ubound1>,[<lbound2>:<ubound2>]
OF <datatype>
```

2.2.3 Files

- show understanding of why files are needed
- use pseudocode for file handling:

```
OPENFILE <filename> FOR READ/WRITE/APPEND // Open file (understand the
                                  difference between various file modes)
READFILE <filename>,<string> // Read a line of text from the file
WRITEFILE <filename>,<string> // Write a line of text to the file
CLOSEFILE <filename>          // Close file
EOF(<filename>)         // function to test for the end of the file
```

- write program code for simple file handling of a text file, consisting of several lines of text

2.3 Programming

Candidates should be able to:

2.3.1 Programming basics

- write a program in a high-level language (The nature of the language should be procedural and will be chosen by the Centre from the following: Python, Visual Basic (console mode), Pascal/Delphi (console mode))
- implement and write a program from a given design presented as either a program flowchart or pseudocode
- write program statements for:
  - the declaration of variables and constants
  - the assignment of values to variables and constants
  - expressions involving any of the arithmetic or logical operators
  - input from the keyboard and output to the console

  (given pseudocode will use the following structures:

```
DECLARE <identifier> : <data type> // declaration
CONSTANT <identifier> = <value>
<identifier> ← <value> or <expression> // assignment)
INPUT <identifier>
OUTPUT <string>
OUTPUT <identifier(s)>
```

2.3.2 Transferable skills

- recognise the basic control structures in a high-level language other than the one chosen to be studied in depth
- appreciate that program coding is a transferable skill

2.3.3 Selection

- use an 'IF' structure including the 'ELSE' clause and nested IF statements
  - given pseudocode will use the following structure:

```
IF <condition>
  THEN
    <statement(s)>
ENDIF
```

  - or, including an 'else' clause:

```
IF <condition>
  THEN
    <statement(s)>
  ELSE
  <statement(s)>
ENDIF
```

- use a 'CASE' structure
  - given pseudocode will use the following structure:

```
CASE OF <identifier>
  <value 1>: <statement>
  <value 2>: <Statement>
  ...
ENDCASE
```

  - alternatively:

```
CASE OF <identifier>
  <value 1>: <statement>
  <value 2>: <Statement>
  ...
  OTHERWISE <statement>
ENDCASE
```

2.3.4 Iteration

- use a 'count controlled' loop:
  - given pseudocode will use the following structure:

```
FOR <identifier> ← <value1> TO <value2>
  <statement(s)>
ENDFOR
```

  - alternatively:

```
FOR <identifier> ← <value1> TO <value2> STEP <value3>
  <statement(s)>
ENDFOR
```

- use a 'post-condition' loop:
  - given pseudocode will use the following structure:

```
REPEAT
  <statement(s)>
UNTIL <condition>
```

- use a 'pre-condition' loop
  - given pseudocode will use the following structure:

```
WHILE <condition>
  <statement(s)>
ENDWHILE
```

- justify why one loop structure may be better suited to a problem than the others

2.3.5 Built-in functions

- use a subset of the built-in functions and library routines supported by the chosen programming language. This should include those used for:
  - string/character manipulation
  - formatting of numbers
  - random number generator
- use the information provided in technical documentation describing functions/procedures

2.3.6 Structured programming

- use a procedure
- explain where in the construction of an algorithm it would be appropriate to use a procedure
  - given pseudocode will use the following structure for procedure definitions:

```
PROCEDURE <identifier>
  <statement(s)>
ENDPROCEDURE
```

  - a procedure may have none, one or more parameters
  - a parameter can be passed by reference or by value
- show understanding of passing parameters by reference

```
PROCEDURE <identifier> (BYREF <identifier>: <datatype>)
  <statement(s)>
ENDPROCEDURE
```

- show understanding of passing parameters by value

```
PROCEDURE <identifier> (BYVALUE <identifier>: <datatype>)
  <statement(s)>
ENDPROCEDURE
```

  - a call is made to the procedure using `CALL <identifier> ()`
- use a function
- explain where in the construction of an algorithm it is appropriate to use a function
- use the terminology associated with procedures and functions: procedure/function header, procedure/function interface, parameter, argument, return value
  - given pseudocode will use the following structure for function definitions:

```
FUNCTION <identifier> RETURNS <data type> // function has no
                                                  parameters
  <statement(s)>
ENDFUNCTION

FUNCTION <identifier> (<identifier>: <data type>)
RETURNS <data type> // function has one or more parameters
  <statement(s)>
ENDFUNCTION
```

  - a function is used in an expression, for example
    - `x ← SQRT(n)`
    - `WHILE NOT EOF(<filename>)`
- write programs containing several components and showing good use of resources

2.4 Software development

Candidates should be able to:

2.4.1 Programming

- show understanding of the design, coding and testing stages in the program development cycle
- show understanding of how to write, translate, test and run a high-level language program
- describe features found in a typical Integrated Development Environment (IDE):
  - for coding, including context-sensitive prompts
  - for initial error detection, including dynamic syntax checks
  - for presentation, including prettyprint, expand and collapse code blocks
  - for debugging, including: single stepping, breakpoints, variables/expressions report window

2.4.2 Program testing

- show understanding of ways of exposing faults in programs and ways of avoiding faults
- locate and identify the different types of errors:
  - syntax errors
  - logic errors
  - run-time errors
- correct identified errors

2.4.3 Testing strategies

- choose suitable data for black-box testing
- choose suitable data for white-box testing
- understand the need for stub testing

Section 3 Advanced Theory

3.1 Data representation

Candidates should be able to:

3.1.1 User-defined data types

- show understanding of why user-defined types are necessary
- define and use non-composite types: enumerated, pointer
- define and use composite data types: set, record and class/object
- choose and design an appropriate user-defined data type for a given problem

3.1.2 File organisation and access

- show understanding of methods of file organisation: serial, sequential (using a key field) and random (using a record key)
- show understanding of methods of file access:
  - sequential access for serial and sequential files
  - direct access for sequential and random files
- select an appropriate method of file organisation and file access for a given problem

3.1.3 Real numbers and normalised floating-point representation

- describe the format of binary floating-point real numbers
- convert binary floating-point real numbers into denary and vice versa
- normalise floating-point numbers
- show understanding of the reasons for normalisation
- show understanding of the effects of changing the allocation of bits to mantissa and exponent in a floating-point representation
- show understanding of how underflow and overflow can occur
- show understanding of the consequences of a binary representation only being an approximation to the real number it represents (in certain cases)
- show understanding that binary representations can give rise to rounding errors

3.2 Communication and Internet technologies

Candidates should be able to:

3.2.1 Protocols

- show understanding of why a protocol is essential for communication between computers
- show understanding of how protocol implementation can be viewed as a stack, where each layer has its own functionality
- show understanding of the function of each layer of the TCP/IP protocol suite
- show understanding of the application of the TCP/IP protocol suite when a message is sent from one host to another on the Internet
- show understanding of how the BitTorrent protocol provides peer-to-peer file sharing
- show an awareness of other protocols (HTTP, FTP, POP3, SMTP) and their purposes

3.2.2 Circuit switching, packet switching and routers

- show understanding of circuit switching and where it is applicable
- show understanding of packet switching
- show understanding of the function of a router in packet switching
- explain how packet switching is used to pass messages across a network, including the Internet

3.2.3 Local Area Networks (LAN)

- show understanding of a bus topology network and the implications of how packets are transmitted between two hosts
- show understanding of a star topology network and the implications of how packets are transmitted between two hosts
- show understanding of a wireless network
- explain how hardware is used to support a LAN: switch, router, servers, Network Interface Cards (NICs), wireless access points
- show understanding of Ethernet and how collision detection and avoidance (such as CSMA/CD) works

3.3 Hardware

Candidates should be able to:

3.3.1 Logic gates and circuit design

- produce truth tables for common logic circuits including half adders and full adders
- derive a truth table for a given logic circuit

3.3.2 Boolean algebra

- show understanding of Boolean algebra
- show understanding of De Morgan's Laws
- perform Boolean algebra using De Morgan's Laws
- simplify a logic circuit/expression using Boolean algebra

3.3.3 Karnaugh Maps

- show understanding of Karnaugh Maps
- show understanding of the benefits of using Karnaugh Maps
- solve binary logic problems using Karnaugh Maps

3.3.4 Flip-flops

- show understanding of how to construct a flip-flop (SR and JK)
- describe the role of flip-flops as data storage elements

3.3.5 RISC processors

- show understanding of the differences between RISC and CISC processors
- show understanding of the importance/use of pipelining and registers in RISC processors
- show understanding of interrupt handling on CISC and RISC processors

3.3.6 Parallel processing

- show awareness of the four basic computer architectures: SISD, SIMD, MISD, MIMD
- show awareness of the characteristics of massively parallel computers

3.4 System software

Candidates should be able to:

3.4.1 Purposes of an operating system (OS)

- show understanding of how an OS can maximise the use of resources
- describe the ways in which the user interface hides the complexities of the hardware from the user
- show understanding of processor management: multitasking, including:
  - the concept of multitasking and a process
  - the process states: running, ready and blocked
  - the need for scheduling
  - the concept of an interrupt
  - how the kernel of the OS acts as the interrupt handler and how interrupt handling is used to manage low-level scheduling
- show understanding of paging for memory management: including:
  - the concepts of paging and virtual memory
  - the need for paging
  - how pages can be replaced
  - how disk thrashing can occur

3.4.2 Virtual machine

- show understanding of the concept of a virtual machine
- give examples of the role of virtual machines
- show understanding of the benefits and limitations of virtual machines

3.4.3 Translation software

- show understanding of how an interpreter can execute programs without producing a translated version
- show understanding of the various stages in the compilation of a program: lexical analysis, syntax analysis, code generation and optimisation
- show understanding of how the grammar of a language can be expressed using syntax diagrams or Backus-Naur Form (BNF) notation
- show understanding of how Reverse Polish Notation (RPN) can be used to carry out the evaluation of expressions

3.5 Security

Candidates should be able to:

3.5.1 Asymmetric keys and encryption methods

- show understanding of the terms: public key, private key, plain text, cipher text, encryption and asymmetric key cryptography
- show understanding of how the keys can be used to send a private message from the public to an individual/organisation
- show understanding of how the keys can be used to send a verified message to the public

3.5.2 Digital signatures and digital certificates

- show understanding of how a digital certificate is acquired
- show understanding of how a digital certificate is used to produce digital signatures

3.5.3 Encryption protocols

- show awareness of the purpose of Secure Socket Layer (SSL)/Transport Layer Security (TLS)
- show awareness of the use of SSL/TLS in client-server communication
- show awareness of situations where the use of SSL/TLS would be appropriate

3.5.4 Malware

- show understanding of malware: viruses, worms, spam, phishing, pharming
- describe vulnerabilities that the various types of malware can exploit
- describe methods that can be used to restrict the effect of malware

3.6 Monitoring and control systems

Candidates should be able to:

3.6.1 Overview of monitoring and control systems

- show understanding of the difference between a monitoring system and a control system
- show understanding of the hardware (sensors, actuators) that are required to build these systems
- show understanding of the software requirements of these systems
- show understanding of the importance of feedback in a control system

3.6.2 Bit manipulation to monitor and control devices

- show understanding of how bit manipulation can be used to monitor/control a device
- carry out bit manipulation operations: test a bit and set a bit (using bit masking) using the instructions from Section 1.4.3 and those listed below
- show understanding of how to make use of appropriate bit manipulation in monitoring systems and control systems

Instruction		Explanation
Op Code	**Operand**	
AND	#n	Bitwise AND operation of the contents of ACC with the operand
AND	<address>	Bitwise AND operation of the contents of ACC with the contents of <address>
XOR	#n	Bitwise XOR operation of the contents of ACC with the operand
XOR	<address>	Bitwise XOR operation of the contents of ACC with the contents of <address>
OR	#n	Bitwise OR operation of the contents of ACC with the operand
OR	<address>	Bitwise OR operation of the contents of ACC with the contents of <address> <address> can be an absolute address or a symbolic address
label>: <op code> <label>:	<operand> <data>	labels an instruction gives a symbolic address <label> to the memory location

Section 4 Further problem-solving and programming skills

4.1 Computational thinking and problem-solving

Candidates should be able to:

4.1.1 Abstraction

- show understanding of how to model a complex system by only including essential details, using:
  - functions and procedures with suitable parameters (as in imperative programming, see Section 2.3)
  - ADTs (see Section 4.1.3)
  - classes (as used in object-oriented programming, see Section 4.3.1)
  - facts, rules (as in declarative programming, see Section 4.3.1)

4.1.2 Algorithms

- write a binary search algorithm to solve a particular problem
- show understanding of the conditions necessary for the use of a binary search
- show understanding of how the performance of a binary search varies according to the number of data items
- write an algorithm to implement an insertion sort
- write an algorithm to implement a bubble sort
- show understanding that performance of a sort routine may depend on the initial order of the data and the number of data items
- write algorithms to find an item in each of the following: linked list, binary tree, hash table
- write algorithms to insert an item into each of the following: stack, queue, linked list, binary tree, hash table
- write algorithms to delete an item from each of the following: stack, queue, linked list
- show understanding that different algorithms which perform the same task can be compared by using criteria such as time taken to complete the task and memory used

4.1.3 Abstract Data Types (ADT)

- show understanding that an ADT is a collection of data and a set of operations on those data
- show understanding that data structures not available as built-in types in a particular programming language need to be constructed from those data structures which are built-in within the language

```
TYPE <identifier1>
   DECLARE <identifier2> : <data type>
   DECLARE <identifier3> : <data type>
   …
ENDTYPE
```

- show how it is possible for ADTs to be implemented from another ADT
- describe the following ADTs and demonstrate how they can be implemented from appropriate built-in types or other ADTs: stack, queue, linked list, dictionary, binary tree

4.1.4 Recursion

- show understanding of the essential features of recursion
- show understanding of how recursion is expressed in a programming language
- trace recursive algorithms
- write recursive algorithms
- show understanding of when the use of recursion is beneficial
- show awareness of what a compiler has to do to implement recursion in a programming language

4.2 Algorithm design methods

Candidates should be able to:

4.2.1 Decision tables

- describe the purpose of a decision table
- construct a decision table for a given problem with a maximum of three conditions
- simplify a decision table by removing redundancies

4.2.2 Jackson Structured Programming (JSP)

- construct a JSP structure diagram showing repetition
- construct a JSP structure diagram showing selection
- write equivalent pseudocode from such structure charts
- construct a JSP structure diagram to describe a data structure

4.2.3 State-transition diagrams

- use state-transition diagrams to document an algorithm
- use state-transition diagrams to show the behaviour of an object

4.3 Further programming

Candidates should already have practical experience of the content in Section 2.3 Programming. Candidates should be able to:

4.3.1 Programming paradigms

- show understanding of what is meant by a programming paradigm
- show understanding of the characteristics of a number of programming paradigms (low-level, imperative, object-oriented, declarative)
  - low-level programming
    - demonstrate an ability to write low-level code that uses various address modes: immediate, direct, indirect, indexed and relative (see Section 1.4.3 and Section 3.6.2)
  - imperative programming
    - see details in Section 2.3
  - object-oriented programming (OOP)
    - demonstrate an ability to solve a problem by designing appropriate classes
    - demonstrate an ability to write code that demonstrates the use of classes, inheritance, polymorphism and containment (aggregation)
  - declarative programming
    - demonstrate an ability to solve a problem by writing appropriate facts and rules based on supplied information
    - demonstrate an ability to write code that can satisfy a goal using facts and rules

4.3.2 File processing (see also Section 2.2.3)

- write code to define a record structure
- write code to perform file-processing operations: open or close a file; read or write a record to a file
- use pseudocode for random file handling:

```
OPENFILE <filename> FOR RANDOM
SEEK <filename>,<address> // move a pointer to the disk address for the record
GETRECORD <filename>,<identifier>
PUTRECORD <filename>,<identifier>
```

- write code to perform file-processing operations on serial, sequential and random files

4.3.3 Exception handling

- show understanding of an exception and the importance of exception handling
- show understanding of when it is appropriate to use exception handling
- write code to use exception handling in practical programming

4.3.4 Use of development tools/programming environments

- describe features in editors that benefit programming
- know when to use compilers and interpreters
- describe facilities available in debuggers and how and when they should be deployed

4.4 Software development

Candidates should be able to:

4.4.1 Software development resources

- show understanding of the possible role of program generators and program libraries in the development process

4.4.2 Testing

- show awareness of why errors occur
- show understanding of how testing can expose possible errors
- appreciate the significance of testing throughout software development
- show understanding of the methods of testing available: dry run, walkthrough, white-box, black-box, integration, alpha, beta, acceptance
- show understanding of the need for a test strategy and test plan and their likely contents
- choose appropriate test data (normal, abnormal and extreme/boundary) for a test plan

4.4.3 Project management

- show understanding that large developments will involve teams
- show understanding of the need for project management
- show understanding of project planning techniques including the use of GANTT and Program Evaluation Review Technique (PERT) charts
- describe the information that GANTT and PERT charts provide
- construct and edit GANTT and PERT charts

7. Description of components

7.1 Scheme of assessment

Each examination paper will consist of a variable number of short-answer and structured questions of variable mark value. Candidates must answer all questions. Candidates will answer on the question paper.

Calculators are **not** allowed in these papers.

Paper 1 Theory Fundamentals
This is a compulsory paper consisting of questions set on Section 1 of the syllabus.

Paper 2 Fundamental Problem-solving and Programming Skills
This is a compulsory paper consisting of questions set on Section 2 of the syllabus.

Paper 3 Advanced Theory
This is a compulsory paper consisting of questions set on Section 3 of the syllabus.

Paper 4 Further Problem-solving and Programming Skills
This is a compulsory paper consisting of questions set on Section 4 of the syllabus.

The assessment is by written papers, but the learning should be done in a mainly practical way: problem-solving and programming. Questions will require the candidate to think, use knowledge with understanding and demonstrate understanding gained through practising practical skills.

7.2 Paper 2 and Paper 4 Problem-solving and Programming Skills pre-release material

The pre-release material for Paper 2 and Paper 4 will be made available to Centres the January before the June examination and the July before the November examination. Centres are advised to encourage their candidates to develop solutions to tasks using a high-level programming language. Centres must choose a high-level programming language from this list: Visual Basic (console mode), Pascal/Delphi (console mode) or Python. The purpose of the pre-release material tasks is to direct candidates to some of the topics which will be examined in Paper 2 and Paper 4. Teachers are expected to incorporate these tasks into their lessons and give support in finding methods and reaching solutions. Questions will be included that test candidates' understanding gained from developing programmed solutions to these tasks. The tasks will be appropriate for all ability levels.

The examination questions will require candidates to have practical programming experience, including writing their own programs, executing (running), testing and debugging them. Candidates are to be encouraged to extend their practical programming beyond the scope of these tasks.It is appreciated that in an examination, candidates will not have access to a compiler, interpreter or any other aid to writing correct syntax. Therefore, minor syntax errors in candidates' programs will be ignored.

8. Notes for the guidance of teachers

Introduction

The purpose of these notes is to provide assistance for teachers preparing candidates for the Cambridge AS and A Level Computer Science examination. They contain notes on equipment, facilities and resources and sources of further information.

Equipment and facilities

Computer science is a practical subject and the Cambridge AS and A Level syllabus places emphasis on the use of procedural high-level programming languages. Centres must ensure that their equipment and facilities are adequate for candidates to be able to satisfy the requirements of the syllabus. The hardware facilities needed will depend on the number of candidates, but should be sufficient for all candidates to have enough time to practise their programming skills.

Hardware

Candidates need to have access to a system with direct-access file capability on backing store and hardcopy facilities.

Software

Candidates should have experience of using a high-level programming language (Pascal/Delphi, Visual Basic or Python), chosen by the Centre.

Books

Provision of textbooks is difficult as new titles are available all the time. The British Computer Society (BCS) book list for schools and colleges lists books which are suitable for use as reference books. Teachers will need to consult several books to cover the whole syllabus adequately. There is a suggested book list on our website. Many schools prefer to have a wide range of reference books rather than a class textbook.

Practical skills

Computing is a practical subject and a range of practical exercises should supplement the study of most parts of the syllabus.

It is important that Centres encourage candidates, as early as possible in the course, to develop a systematic approach to practical problem-solving using appropriate resources.

9. Other information

Equality and inclusion

Cambridge International Examinations has taken great care in the preparation of this syllabus and assessment materials to avoid bias of any kind. To comply with the UK Equality Act (2010), Cambridge has designed this qualification with the aim of avoiding direct and indirect discrimination.

The standard assessment arrangements may present unnecessary barriers for candidates with disabilities or learning difficulties. Arrangements can be put in place for these candidates to enable them to access the assessments and receive recognition of their attainment. Access arrangements will not be agreed if they give candidates an unfair advantage over others or if they compromise the standards being assessed.

Candidates who are unable to access the assessment of any component may be eligible to receive an award based on the parts of the assessment they have taken.

Information on access arrangements is found in the *Cambridge Handbook* which can be downloaded from the website **www.cie.org.uk/examsofficers**

Language

This syllabus and the associated assessment materials are available in English only.

Grading and reporting

Cambridge International A Level results are shown by one of the grades A*, A, B, C, D or E, indicating the standard achieved, A* being the highest and E the lowest. 'Ungraded' indicates that the candidate's performance fell short of the standard required for grade E. 'Ungraded' will be reported on the statement of results but not on the certificate. The letters Q (result pending), X (no results) and Y (to be issued) may also appear on the statement of results but not on the certificate.

Cambridge International AS Level results are shown by one of the grades a, b, c, d or e, indicating the standard achieved, 'a' being the highest and 'e' the lowest. 'Ungraded' indicates that the candidate's performance fell short of the standard required for grade 'e'. 'Ungraded' will be reported on the statement of results but not on the certificate. The letters Q (result pending), X (no results) and Y (to be issued) may also appear on the statement of results but not on the certificate.

If a candidate takes a Cambridge International A Level and fails to achieve grade E or higher, a Cambridge International AS Level grade will be awarded if both of the following apply:

- the components taken for the Cambridge International A Level by the candidate in that series included all the components making up a Cambridge International AS Level
- the candidate's performance on these components was sufficient to merit the award of a Cambridge International AS Level grade.

For languages other than English, Cambridge also reports separate speaking endorsement grades (Distinction, Merit and Pass), for candidates who satisfy the conditions stated in the syllabus.

Entry codes

To maintain the security of our examinations, we produce question papers for different areas of the world, known as 'administrative zones'. Where the component entry code has two digits, the first digit is the component number given in the syllabus. The second digit is the location code, specific to an administrative zone. Information about entry codes for your administrative zone can be found in the *Cambridge Guide to Making Entries*.

Cambridge International Examinations
1 Hills Road, Cambridge, CB1 2EU, United Kingdom
Tel: +44 (0)1223 553554 Fax: +44 (0)1223 553558
Email: info@cie.org.uk www.cie.org.uk