GCSE Computer Science Booster Pack

Commissioned by The PiXL Club Ltd.

Welcome to the GCSE Computer Science Booster Pack. The aim of this pack is to help support the teaching of students of computer science who are also foundation tier maths students.
The intention of the pack is to provide a simple and accessible explanation of the mathematical aspects of the computer science GCSE. These explanations are followed with a variety of questions and tasks.

This could be applied to teaching in a number of ways.
E.g. to support identified students prior to whole class teaching, as a therapy tool in the DTT process.

Notes for use:

- **The layout of the document is like a textbook. There is no space for pupils to write answers**
- **There is an answer booklet to support the delivery of the pack's content**
- **There are many instances of pseudocode in the pack. Where this is the case the pseudocode used has followed OCR and AQA's pseudocode guidance**
- **However, each exam board uses varying pseudocode conventions. Please refer to your specific exam board's guide wherever possible.**

Contents

Relational Operators

Relational operators, sometimes known as comparison operators, test the relationship between two numbers. This can include whether they are equal to each other, whether one number is greater etc.

< is a relational operation. In maths is also called an inequality sign.

< means a number that is less than another number

Example

4 is *less than* 12 can be written as 4 < 12

> is also a relational operation.

> means a number that is greater than another number

Example

40 is *greater than* 12 can be written as 40 > 12

Other relational operators which you will be need to know and be able to use are:

- == which means equal to
- ≤ or <= which means *less than or equal to*
- ≥ or >= which means *greater than or equal to*
- ≠ or != which means is not equal to

Exercise A

1 Which of the following are TRUE and which are FALSE? If your answer is FALSE give the correct relational operation choosing from < and >.

a 3 < 10 b 5 > 12 c 8 > 2 d 10 < 45 e 17 < 2 + 3

f 20 - 5 > 16 g 30 ÷ 5 > 4 h 12 > 2 * 5 i 20 - 3 > 2 * 8 j 2 + 5 + 7 > 20 - 4 – 1

2 Copy each question writing the correct relational operation for each. Choose from ≤ and >.

a 3 ◯ 6 b 10 ◯ 8 c 25 ◯ 27 d -8 ◯ 8

e 13 ◯ 62 f 42 ◯ 6 g 19 ◯ -29 h 0 ◯ -5

3 Copy each question writing the correct relational operation for each. Choose from ≤ and >.

a 10 ◯ 7 + 2 b 25 - 10 ◯ 8 + 8 c 2 * 3 ◯ 1 + 6

d 25 ◯ 48 – 25 e -5 ◯ 8 – 14 f -10 ◯ 46 – 47

g -11 ◯ -3 - 9 h 22 / 2 ◯ 3 * 4 i 11 / 2 ◯ 7 * 5

4 Which of the following are TRUE and which are FALSE? If your answer is FALSE give the correct relational operation choosing from == and ≠.

a 10 == 10 b 14 != 7 * 2 c 13 == 31 – 17

d 23 == 48 – 25 e -5 != 9 – 17 f 23-30 == -6

g -7 == -3 - 7 h 17 == 51 / 3 i -1 != 21 / 3 - 6

5 Copy each question writing the correct relational operation for each. Choose from == and ≠

a 10 ◯ 7 + 3 b 16 ◯ 48 / 3 c 24 ◯ 6 * 4

d 18 ◯ 34 – 25 e -11 ◯ 8 – 14 f -10 ◯ 33 – 43

g -9 ◯ 4 - 14 h 13 ◯ 52 / 4 i 22 + 26 ◯ 12 * 4

6 Which of the following are TRUE and which are FALSE? If your answer is FALSE give the correct relational operation choosing from ≤ and >.

a 10 ≥ 7 + 2 b 22 ≥ 12 + 9 c 21 ≤ 7 * 4

d 19 ≥ 43 – 25 e -5 ≤ 8 – 14 f -6 ≤ 5 - 8

g 7 + 3 ≤ 33 / 3 h 15 ≥ 34 / 2 i 36 ≥ -12 * 3

7 Copy each question writing the correct relational operation for each. Choose from < and ≥.

a 19 ◯ 11 + 7 b 20 - 4 ◯ 12 + 6 c 13 ◯ 14 – 17

d 7 ◯ 32 – 25 e -5 ◯ 12 – 14 f -10 ◯ 46 – 47

g -9 ◯ -3 -9 h 16 ◯ 80 / 5 i 48 ◯ 12 * 3

Arithmetic Operators

Addition, Subtraction, Multiplication and Division.

Arithmetic operators are used to perform a calculation, just like they are in conventional mathematics.

Due to the symbols that are available on a computer, the symbols differ slightly to the ones that you are familiar with.

Symbol	Name	Example
+	Addition (+)	`print 6 + 2` `>>> 8`
-	Subtraction (-)	`print = 3 – 2` `>>> 1`
*	Multiplication (x)	`print = 5 * 2` `>>> 1`
/	Division (÷)	`print = 16 / 8` `>>> 2`

Arithmetic rules apply for the **order of operations** in Computer Science:

Brackets	()	
Order	^	
Division	÷	/
Multiplication	x	*
Addition	+	
Subtraction	–	

Example

Calculate 24 / 3 + 9

Work out 24 / 3 first

Answer: 8+9 = 17

Exercise A

1 Work out the correct answer to each calculation.

a 55 + 20 = b 5 * 12 = c 42 - 13 = d 45 / 9 =

e 18 * 2 / 3 = f 20 * 5 - 16 = g 30 / 5 + 4 = h 124 / 2 =

i 66 - 6 / 3 = j 2 + 5 * 7 =

2 Copy each question writing the correct arithmetic operation for each.

Choose from +, -, * or /

a 6 ◯ 6 = 36 b 80 ◯ 8 = 10 c 25 ◯ 27 = 52 d -8 ◯ 8 = 0

e 13 ◯ 49 = 62 f 42 ◯ 2 = 21 g 15 ◯ 4 = 60 h 0 ◯ 5 = -5

Division with Remainders

When we divide one number by another we can write the answer in lots of different ways.

For example

21 / 4 = 5.25

We could also write

21 / 4 = 5 ¼

You might also have seen division with remainders

21 / 4 = 5 remainder 1

4 goes into 21 5 whole times, with 1 left over

The whole number part is called the quotient and the number left over is the remainder

For larger numbers you can use a method such as short division to find the answer with a remainder

Example

Work out 266 / 3

$$\begin{array}{r} 0\ \ 8\ \ \ 8\ \ r\ \ 2 \\ 3\overline{)\ \cancel{2}\ \ {}^{2}6\ \ \ {}^{2}6\ \ } \end{array}$$

Answer: 88 remainder 2

(Check that 266 = 88 * 3 +2)

Exercise B

1 Work out each of the following. Give your answers as a whole number with a remainder. The first one is done for you.

a 37/4 = 9 r 1 b 55/3 c 60/2 d 48/5 e 82/3 f 77/2

g 100/4 h 62/3 i 64/3 j 82/5 k 49/4 l 88/9

2 Work out each of the following. Give your answer with a remainder.

a 116/5 b 121/7 c 111/2 d 148/5 e 157/3 f 166/8

g 119/4 h 232/7 i 199/6 j 187/6 k 195/4 l 255/9

Modulo

In Computer Science, the **modulo** operation finds *only* the remainder when one number divided is by another. This operation is often written as **MOD or %.**

Example

Find 7 MOD 2 *We would say "7 modulo 2"*

7 / 2 = 3 remainder 1

So 7 MOD 2 = 1

This operation would result in the answer 1 because 5 divided by 2 leaves 1 as a remainder.

Example

Find 9 % 3 *We would say "9 modulo 3"*

9 / 3 = 3 r 0

So 9 % 3 = 0

This operation would result in the answer 0 because 9 divided by 3 divides leaving no remainder.

Exercise C

1 Calculate each of the following.

a 22 MOD 4 b 38 MOD 3 c 50 MOD 4 d 55 MOD 6 e 82 MOD 3 f 48 MOD 6

g 110 MOD 3 h 99 mod 8 i 64 MOD 5 j 80 MOD 5 k 51 MOD 7 l 88 MOD 6

2 Calculate each of the following.

a 112 % 7 b 118 % 8 c 120 % 8 d 143 % 5 e 156 % 9 f 162 % 8

g 113 % 11 h 219 % 7 i 203 % 6 j 181 % 6 k 199 % 4 l 243 % 8

Quotient

In Computer Science, the **DIV** operation finds *only* the whole number answer when one number divided is by another. (Remember this is called the quotient.) This operation is often written as **DIV or //**. It is also called *integer division.*

Example

Find 7 DIV 3

7 / 3 = 2 r 1

So the answer is 2 because 3 goes into 7 2 whole times and we ignore the remainder

7 DIV 3 = 2

Example

Find 12 // 4

12 / 4 = 3 r 0

So the answer is 3 because 4 goes into 12 3 whole times.

Together, DIV and MOD allow you to find the quotient and remainder in a division

For example

14 / 3 = 4 r 2

Quotient = 4 14 DIV 3 = 4

Remainder = 2 14 MOD 3 = 2

Exercise D

1 Work out each of the following

a 23 DIV 3 b 36 DIV 5 c 51 DIV 4 d 30 DIV 6 e 44 DIV 3 f 54 DIV 6

g 99 DIV 7 h 42 DIV 8 i 75 DIV 4 j 81 DIV 9 k 63 DIV 4 l 87 DIV 6

2 Work out each of the following

a 112 // 6 b 111 // 5 c 138 // 7 d 140 // 4 e 156 // 4 f 173 // 8

g 130 // 7 h 240 // 3 i 207 // 5 j 189 // 6 k 193 // 4 l 251 // 6

3 Match the calculations, quotients (DIV) and remainder (%)

15 / 4	6	3
29 / 5	8	2
36 / 6	5	1
34 / 16	3	0
25 / 3	2	4

Number Systems and Conversions

Denary Number System (Decimal)

The **denary** (**decimal**) number system is based on 10 numbers.

The number system used in the UK is denary (decimal) and uses the digits from 0 – 9.

It uses 'places' to determine the value of a number according to the column in which the digits are written.

For example, the digit 2 is worth 20 in the number 26 and the digit 2 is worth 2000 in the number 2067. Both digits are the same but their value is changed by their place.

This is known as **place value**.

Thousands	Hundreds	Tens	Units
		2	6

Thousands	Hundreds	Tens	Units
2	0	6	7

The value of a digit in each column is 10 times (x10) larger than the same digit in the column to its right. It is 10 times smaller than the same digit in the column on its left. This is because the decimal (denary) system is **base 10**.

Examples

- *80 is ten times larger than 8.*
- *800 is one hundred times larger than 8.*
- *60 is ten times smaller than 600.*

Exercise A

1 Write down the value of the digit 3 in each number.

A 23 b 130 c 3400 d 173 e 7370

f 631 g 13100 h 3 900 000 I 5 301 100 j 2 453 200

2 For each question decide whether the statement is TRUE or FALSE.

If FALSE write the correct statement.

a 4000 is one thousand times larger than 4

b 700 is one hundred times larger than 7

c 900 is one hundred times larger than 90

d 20 000 is one hundred times larger than 200

e 5 is ten times smaller than 50

f 60 is one thousand times smaller than 6 000

g 2 000 000 is one thousand times larger than 2 000

h 800 is one thousand times smaller than 80 000

3 For each pair of numbers write down how many times larger the first is than the second.

a 60 : 6	b 300 : 30	c 2000 : 200
d 600 : 6	e 4000 : 4	f 90 000 : 9 000
g 80 000 : 800	h 50 000 : 500	i 700 000 : 7 000

4 For each pair of numbers write down how many times smaller the first is than the second.

a 7 : 700	b 60 : 600	c 8 : 8 000
d 2 : 20 000	e 400 : 4 000	f 90 : 90 000
g 700 : 70 000	h 300 : 30 000	i 5 000 : 5 000 000

5 Write these numbers in order of size, smallest first.

7000000 70 7000 7 700 70000

Thousands Hundreds Tens Units

The column headings can be written as **powers** of 10 like this:

10^3 10^2 10^1 10^0

The value of the power shows how many 'times 10' the digit in each column is worth.

Examples

10^5	10^4	10^3	10^2	10^1	10^0	*or*	100000s	10000s	1000s	100s	10s	1s
	3	5	2	7	9			3	5	2	7	9

The number 35 279 is made up of 5 digits.

In this number 9 is worth 9 x 1 =9

7 is worth 7 x 10 = 70

2 is worth 2 x 100 = 200

5 is worth 5 x 1000 = 5 000

3 is worth 3 x 10 000 = 30

6 Write down the values of these numbers in full:

a 10^2 b 10^4 c 10^0 d 10^5 e 10^1 f 10^3

7 Write these numbers as powers of 10.

a 1 000 b 1 c 100 d 10 000 e 100 000 f 1 000 000

8 Match the numbers written in full to the numbers written as powers of 10.

Find the odd one out.

10 000 100 1 000 1 000 000 10 100000

10^5 10^3 10^1 10^0 10^4 10^2 10^6

Binary Number System

The **binary** number system is based on 2 numbers.

The binary system only uses the digits 0 and 1.

The digits 2, 3, 4, 5, 6, 7, 8 and 9 do not exist in the binary system.

The binary system uses place value in the same way as the denary (decimal) system.

Compare denary (decimal) and binary headings:

	X10	x10	x10	
Denary (decimal)	Thousands	Hundreds	Tens	Units
	X2	x2	x2	
Binary	Eights	Fours	Twos	Units

Compare denary (decimal) and binary column headings as powers:

Denary (decimal)	10^3	10^2	10^1	10^0
Binary	2^3	2^2	2^1	2^0
Or	8	4	2	1

Examples of binary numbers:

- 10
- 1001
- 111
- 100100

The value of the power shows how many 'times 2' the digit in each column is worth. The binary number system is referred to as **base 2** and the decimal (denary) system is base 10.

Examples

32	16	8	4	2	1
	1	*0*	*0*	*1*	*1*

OR

32	16	8	4	2	1
	1	*0*	*0*	*1*	*1*

In this number *1 is worth 1 x 1 = 1*

1 is worth 1x2 = 2

0 is worth 0x 4 = 0

0 is worth 0 x 8 = 0

1 is worth 1 x 16 = 16

Exercise B

1 For each statement write TRUE or FALSE. Give a reason for your answer.

a 3 210 is a binary number because it has 0 and 1 in it.

b 201 is a binary number because it only has digits up to 2.

c 5565 is not binary because it uses digits 5 and 6 which are not in the binary number system

d 100 is a decimal (denary) number because it is 10 times 10 and the decimal (denary) system is based on powers of 10.

e 1 010 could be binary or decimal (denary).

2 Find the value of the blue and orange 1's in each of these binary numbers.

[Draw a place value grid to help you if you need to]

128	64	32	16	8	4	2	1

a 10 b 1000 c 1000000 d 101 e 1010

f 110 g 1100000 h 100001 i 11000000 j 10 010

3 For each statement write TRUE or FALSE. If your answer is FALSE give the correct answer.

[All the statements relate to the binary system].

a 10 is twice as large as 1.

b 100 is four times as large as 10.

c 1000 is eight times as large as 1.

d 10 is 4 times smaller than 10000.

e 100 is eight times smaller than 100000

f 10000000 is sixteen times larger than 1000.

4 Match the numbers written in full to the numbers written as powers of 2.

Find the odd one out.

16 1 64 32 8 128 2

2^1 2^4 2^6 2^0 2^5 2^2 2^3 2^7

Converting Denary Numbers into Binary Numbers

Here are two methods for converting denary numbers into binary numbers.

Example: *Convert 25 from denary to binary.*

Place value method

Step One: Begin with a binary place value grid.

Start on the left until you find the highest number that can be subtracted from 25 and put a 1 in that place

128	64	32	16	8	4	2	1
			1				

Step Two: Find the difference 25- 16 = 9

Step Three: Repeat the process

128	64	32	16	8	4	2	1
			1	1			

9-8=1

128	64	32	16	8	4	2	1
			1	1			1

1-1=0

Step Four: Put zero's in the spaces

128	64	32	16	8	4	2	1
			1	1	0	0	1

Answer: 25 is 1101 in binary

Division by 2 method:

This method involves remainders and repeatedly dividing by 2.

Number	***Calculation***	***Answer***	***Remainder***
25	*25/2*	*12*	*1*
12	*12/2*	*6*	*0*
6	*6/2*	*3*	*0*
3	*3/2*	*1*	*1*
1	*1/2*	*0*	*1*

Repeat until the answer is 0

REMEMBER: The 1 that you are left with at the bottom is still a remainder and must be included within your answer.

Read the number from bottom to top

Answer

25 = 11 001 in binary

[You can check your answer by converting 11 001 back to denary]

Exercise C

1 Convert each number from denary to binary.

a 10	b 15	c 16	d 24	e 30	f 100
g 212	h 78	I 200	j 146	k 230	i 255

2 Find the answer to each denary addition. Give your answer in binary.

a 10 + 21	b 19 + 30	c 22 + 20	d 7 + 14
e 6 + 11 + 15	f 35 - 15	g 100 - 36	h 45 - 19
i 400 - 312	j 342 - 87		

3 Find the answer to each denary calculation. Give your answer in binary.

a 16 * 4	b 20 * 6	c 7 * 7	d 48 / 4
e 111 + 32	f 100 + 75	g 222 / 2	h 296 - 70

A **bit** is a binary number

0 and 1 are bits

A **nibble** is a set of four binary digits.

1001 is a nibble

A **byte** is a set of 8 binary digits

10001000 is a byte

4 For each number write down bit, nibble or byte

a 1	b 1111	c 11111111	d 1010	e 0	f 10110110

Converting Binary Numbers to Denary Numbers

Numbers that look the same have different values in different number systems because they have a different **base.**

100 in binary is not the same value as 100 in the denary (decimal) system.

The number 10 in the binary system has a different value to the number 10 in the denary (decimal) system**.**

To convert a binary number to a denary (decimal) number, write columns headings as place value.

128 64 32 16 8 4 2 1

Write the binary number underneath and added together the value of the columns to make the conversion.

Example

Convert 10110 from binary to denary (decimal).

Write the number beneath the appropriate column headings:

16	*8*	*4*	*2*	*1*
1	*0*	*1*	*1*	*0*

Add up the column heading for the columns where there is a 1.

16 + 4 + 2 = 22

10110 in the binary system is equivalent to 22 in the denary (decimal) system.

The base of each number can be written like this:

$11110_2 = 22_{10}$

Exercise D

1 Convert the following numbers from binary to denary (decimal).

Show your calculations for each.

a 1110 b 10011 c 1100 d 10100

e 101001 f 110011 g 1011011 h 1000 000

i 1100100 j 11000011

2 Which of the following are TRUE and which are FALSE?

a $1001_2 == 32_{10}$ b $101101_2 > 40_{10}$ c $45_{10} == 101101_2$

d $134_{10} > 1101111_2$ e $93_{10} < 11110_2$ f $110110_2 > 110_2$

g $1110111_2 == 119_{10}$ h $1011101_2 < 102_{10}$ i $100101_2 > 31_{10}$

j $10110101_2 == 181_{10}$

3 Find the answer to each question. Give your answer in denary (decimal).

a $10d_{10} + 11_2$ b $100_2 + 75_{10}$ c $111_2 + 32_{10}$ d $512_{10} + 1000111_2$

e $10_{10} – 10_2$ f $100_{10} – 11000_2$ g $48_{10} – 110000_2$ h $10010011_2 – 70_{10}$

4 Find the answer to each question. Give your answer in denary (decimal)

a $10_2 * 10_2$ b $11_2 * 100_2$ c $10_2 * 1001_2$ d $1000_2 / 10_2$ e $111_2 + 32_{10}$

f $100_2 + 48_{10}$ g $100000_2 / 1000_2$ h $10000_2 - 5_{10}$

5 Use the number sequence below to answer the following questions.

1001 10011 11101 100111

a Convert each number in this sequence from binary to denary (decimal).

b Write down the next number in the sequence in binary and denary (decimal)

Hexadecimal Number System

The **hexadecimal** number system is based on 16 numbers.

The hexadecimal system uses the following digits:

0 1 2 3 4 5 6 7 8 9 A B C D E F

This table shows the numbers to 16 in decimal, hexadecimal and binary.

Decimal	Hexadecimal	Binary
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

As with binary and denary, hexadecimal also referred to by its base.

Hexadecimal is **base 16**.

Hexadecimal is based on powers of 16.

The 8 is worth 8 x 16^0 or 8 x 1

The 5 is worth 5 x 16^1 or 5 x 16

In GCSE Computer Science you will only be working with numbers in the first two columns in hexadecimal.

Converting Hexadecimal Numbers into Binary Numbers

Use a place value grid to convert from hexadecimal to binary.

Write a set of place value headings for each digit.

Place value grids for 96 in hexadecimal looks like this:

			9	6			
8	**4**	**2**	**1**	**8**	**4**	**2**	**1**

Example

Convert 96_{10} into hexadecimal.

Split the number in half and write the first half of the binary scale on each side. This is because you are working with two **nibbles**.

			9	6			
8	4	2	1	8	4	2	1
1	**0**	**0**	**1**	**0**	**1**	**1**	**0**
	8 + 1 = 9				4 + 2 = 6		

You can then find which numbers in the binary scale add up to make the single hexadecimal digit.

Answer

96 in hexadecimal = 10010110 in binary

Example

Convert AF from hexadecimal into binary.

REMEMBER: if the hexadecimal number contains the symbols A-F, you must first convert it to its denary equivalent.

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DEC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

			A	F			
			10	15			
8	4	2	1	8	4	2	1
1	**0**	**1**	**0**	**1**	**1**	**1**	**1**
	8 + 2 = 10			8 + 4 + 2 + 1 = 15			

Find which numbers in the binary scale add up to make the single hexadecimal digit.

Answer

AF = 10101111

NOTE: if the hexadecimal number is only a single digit e.g. A, 9 etc. you would only be required to write the scale once as the answer with only be 4 digits (a nibble)

Exercise E

1 Convert each number from hexadecimal to binary.

a A b F c 14 d 1C e E6 f 9E

g 91 h D9 I D7 j AF k DD l FF

2 Match the hexadecimal number with its binary equivalent

Hexadecimal	B3	16	D6	25	3F

Binary	10110	11010110	100101	111111	10110011

3 Which of the following are TRUE and which are FALSE?

Each question contains one binary number and one hex number

a $16_{16} < 0001\ 0110_2$ b $0011\ 0000_2 > 30_{16}$ c $61_{16} == 0110\ 0001_2$

d $CC_{16} > 0011\ 0101_2$ e $E8_{16} == 1110\ 1101_2$ f $D7_{16} < 1101\ 0110_2$

g $FF_{16} < 1111\ 1110_2$ h $AF_{16} > 1000\ 1111_2$ i $1110\ 1111_2 == EF_{16}$

j $ED_{16} < 1100\ 1110_2$ k $AC_{16} > 1010\ 1111_2$ l $1101\ 0110_2 == DE_{16}$

Converting Binary Numbers into Hexadecimal Numbers

To convert binary to hexadecimal, reverse the process of converting from hexadecimal to binary.

Use a place value grid like this to convert from binary to denary.

8	**4**	**2**	**1**	**8**	**4**	**2**	**1**

Example

Convert 11001011_2 into hexadecimal.

Split the number into two nibbles. Write the first half of the binary scale on each side.

8	4	2	1	8	4	2	1
1	**1**	**0**	**0**	**1**	**0**	**1**	**1**

Add the numbers together to find the total for each side.

8	4	2	1	8	4	2	1
1	**1**	**0**	**0**	**1**	**0**	**1**	**1**
1x8 + 1x4 = 12				1x8 + 1x2 + 1x1 =11			

Convert 12 and 11 into hexadecimal

C B

REMEMBER: if the number is greater than 10 then it needs to be converted into its hexadecimal equivalent first.

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DEC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Answer

11001011 = CB

Converting numbers that are not 8 bits in length

There are two ways to convert numbers that are not 8 bits in length. E.g. 101101

Either

Split the binary number 4 bits from the right-hand side:

2	1	8	4	2	1
1	**0**	**1**	**1**	**0**	**1**
2		8 + 4 + 1 = 13			

Or

Add zeros to the left-hand side until the number is 8 bits in length:

8	4	2	1	8	4	2	1
0	**0**	**1**	**0**	**1**	**1**	**0**	**1**
2				8 + 4 + 1 = 13			

Answer

101101 = 2D

Exercise F

1 Convert each number from binary to hexadecimal.

a 0010 1101	b 0101 0101	c 1011 0010	d 0110 1010
e 1011 1110	f 0011 0101	g 1110 1110	h 1101 0101
i 0111 0100	j 0010 1010	k 0011 1010	l 0111 0111

2 Compare the binary and hexadecimal numbers.

Which of the following are TRUE and which are FALSE?

a $19_{16} < 0011\ 1001_2$	b $0001\ 1101_2 > 25_{16}$	c $64_{16} == 0110\ 0100_2$
d $1010\ 0100_2 < A7_{16}$	e $6C_{16} > 0110\ 1011_2$	f $E9_{16} == 1110\ 1001_2$
g $0111\ 0001_2 < 75_{16}$	h $3D_{16} < 0011\ 1111_2$	i $1111\ 1101_2 > FF_{16}$
j $0110\ 1101_2 == ED_{16}$	k $1111\ 1010_2 == DA_{16}$	l $1101\ 1001_2 == DD_{16}$

3 Which of these binary numbers are less than $B2_{16}$?

0100 0000; 1010 1010; 1100 0011; 1011 1101; 1111 1111;

Converting Hexadecimal Numbers into Denary Numbers

Converting from hexadecimal to denary uses the place value of each number.

This number is 42 in the hexadecimal system 42_{hex}

Example

Convert 93_{hex} to denary

x16

16^1 16^0

9 3

The 3 is worth $3 \times 16^0 = 3 \times 1 = 3$

The 9 is worth $9 \times 16^1 = 9 \times 16 = 144$

Answer

3 + 144 = 147

Example

Find the value of $7F_{hex}$ in denary.

If the hexadecimal number includes letters then you must convert it to its denary equivalent first.

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DEC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

x16

16^1 16^0

7 F

If a letter is used, you must work out the numerical value F = 15

The F is worth $15 \times 16^0 = 15 \times 1 = 15$

The 7 is worth $7 \times 16^1 = 7 \times 16 = 112$

Answer

15 + 112 = 127

Exercise G

1 Convert each number from hexadecimal to denary.

a A b F c 14 d 1C e E6 f 9E

g 91 h D9 i D7 j AF k DD i FF

2 Match the denary workings out with the hexadecimal number

4 x 16 + 15	9D
9 x 16 + 4	D4
9 x16 +13	4F
13 x 16 +4	D9
13 x 16+9	94

3 Which of the following are TRUE and which are FALSE?

a $15_{16} < 21_{10}$ b $19_{16} > 25_{10}$

c $64_{16} == 100_{10}$ d $120_{10} < 7D_{16}$

e $CC_{16} > 200_{10}$ f $E9_{16} == 233_{10}$

g $180_{10} < AB_{16}$ h $FC_{16} < 250_{10}$

i $255_{10} > FF_{16}$ j $239_{10} == EF_{16}$

4 Find the answer to each calculation. Give your answer in <u>denary</u>.

a $16_{16} + 4_{10}$ b $29_{16} * 2_{10}$

c $4C_{16} - 7_{10}$ d $30_{16} / 4_{10}$

e $36_{10} + DE_{16}$ f $CD_{16} + 99_{10}$

g $F9_{16} - 59_{10}$ h $EE_{16} / 2_{10}$

5 Write these numbers in order of size, smallest first.

a AB_{16} 25_{10} $1F_{16}$ 300_{10} 99_{16}

b $C3_{16}$ $9F_{16}$ 200_{10} $4B_{16}$ 80_{10}

Converting Denary Numbers into Hexadecimal Numbers

Here are two methods for converting denary numbers into hexadecimal numbers

Example

Convert 199 from denary to hexadecimal.

Method One

Step One: Convert the denary number to binary.

128	64	32	16	8	4	2	1
1	1	0	0	0	1	1	1

199 = 11000111

Step Two: Convert the binary number into hexadecimal.

8	4	2	1	8	4	2	1
1	**1**	**0**	**0**	**0**	**1**	**1**	**1**
	12				7		

11000111 = C7

Answer: 199 is C7 in hexadecimal

Method Two

Use division to convert from denary to hexadecimal.

Divide the number by 16. Remainders are also used in this conversion.

Example

Convert 199 from denary to hexadecimal.

	0	1	2	r 7
16	~~1~~	~~19~~	39	

REMEMBER: if the number is greater than 9 you must convert it to the hexadecimal equivalent.

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DEC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

12 = C

Answer:

199 = C7

[You can check your answer by converting $C7_{hex}$ back to denary]

Exercise H

1 Convert each number from denary to hexadecimal.

a 9 b 15 c 40 d 90 e 122 f 135

g 157 h 164 i 199 j 217 k 200 i 240

2 Compare these denary and hexadecimal numbers.

Which of the following are TRUE and which are FALSE?

a $2C_{16} == 44_{10}$ b $38_{16} > 56_{10}$ c $78_{16} == 114_{10}$ d $100_{10} < 7E_{16}$

e $B4_{16} > 176_{10}$ f $C9_{16} == 201_{10}$ g $213_{10} < DA_{16}$ h $FA_{16} < 250_{10}$

i $232_{10} > E8_{16}$ j $220_{10} == EC_{16}$

3 Find the answer to each question. Give your answer in <u>hexadecimal</u>.

a $10_{16} + 14_{10}$ b $36_{16} * 2_{10}$ c $3A_{16} - 19_{10}$ d $54_{16} / 4_{10}$

e $EC_{16} - 22_{10}$ f $AC_{16} + 65_{10}$ g $FA_{16} - 44_{10}$ h $CE_{16} / 2_{10}$

4 Use the hexadecimal number sequence below to answer the following questions.

D 17 21 2B

a Find the first four numbers in this sequence in denary.

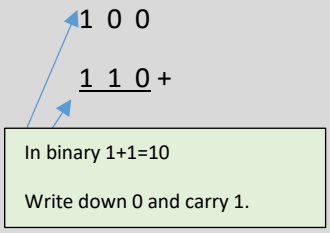
b Write down the next two numbers in the sequence in hexadecimal

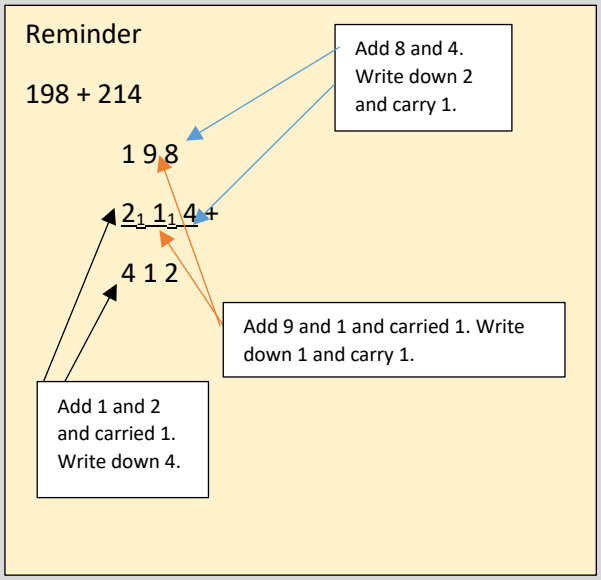
Addition of Binary Numbers

The same method for addition is used in binary as in denary (decimal).

Example

Find 100 + 110





Answer: *100 + 110 = 1011*

In the addition of two binary numbers there are only four different possible outcomes: (denary)

0 + 0 = 0 (0)
1 + 0 = 1 (1)
0 + 1 = 1 (1)
1 + 1 = 10 (2)
1 + 1 + 1 = 11 (3) - Occurs if a number is carried to the next column

Exercise I

1 Work out the answer to each binary sum. Give your answer in binary.

a 10 + 10 b 101 + 100 c 11 + 10

d 100 + 100 e 11 + 100 f 101 + 101

2 Work out the answer to each binary sum. Give your answer in binary.

a 111 + 111 b 1000 + 11 c 1000 + 100

d 1101 + 101 e 101101 + 11101 f 101101 + 11110

g 1101011 + 1001100 h 1110101 + 1111111 i 1010101 + 1101110

3 Find the pairs of binary sums that give the same answer.

1100 + 1101	1010 + 100	1000000 + 100	11001 + 1001011
1000 + 100011	10100 + 101	110010+ 110010	11001 + 10100
10 + 101	100000 + 100000	11 + 100	1 + 1101

4 Jade and Cameron are playing a Higher or Lower Binary Card game. For each round they each turn over a card and win a point if their answer, in denary (decimal), is higher.

Find the value for each card.
Work out which player wins each round. (Convert their total to denary)
Write down who wins the game and their score.

	Jade	**Cameron**
Round 1	10010 + 10001	10000 + 10101
Round 2	11110 + 11011	11010 + 11000
Round 3	110100 + 100111	101111 + 101010
Round 4	111111 + 110000	100110 + 111000
Round 5	1101001 + 1101011	1011000 + 1011100

Adding Three Binary Numbers

Use column addition to add binary numbers.

Adding three binary numbers can result in more than one carry.

Example

Find 10+11+100

4	2	1	
	1	0	
	1	1	
1	0	0	+
1	0	0	1
1			

Add 0 + 1 + 1 = 1 write down 1

Add 1 + 1 + 0 = 10 write down 0, carry 1

Add 1 + 1 = 10 write down 0 carry 1

Answer *10+ 11 + 100 = 1001*

Exercise J

1 Add the following sets of three binary numbers. Give your answer in binary.

a 10 + 101 + 1000

b 1001 + 101 + 1010

c 100100 + 10010 + 1011

d 10111 + 101000 + 111

e 1001 + 1010100 + 101010

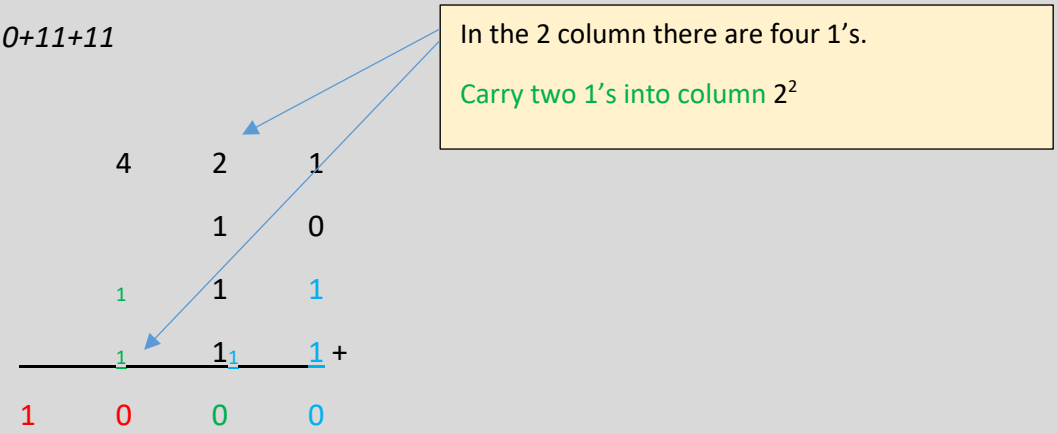
2 For each pair of calculations insert either <, ==, or >.

a 100 + 101 +1001 ◯ 1000 + 10000 +11

b 110011 + 1001 010 + 1011101 ◯ 101010 + 1011011 + 110001

Example

Find 10+11+11



Answer *10 + 11 + 11 = 1000*

3 Add the following sets of three binary numbers. Give your answer in binary.

a 11 + 11 + 11

b 101 + 111 + 101

c 1 011 + 1111 +1101

d 10111 + 11011 + 11001

e 11111 + 1111 + 101111

f 100111 + 110111 + 11011

g 11111 + 11111 + 101101

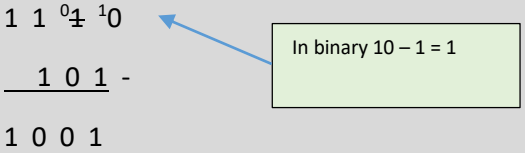
h 1011111 + 110111 + 110101

Subtraction of Binary Numbers

The same method for subtraction is used in binary as in denary (decimal).

Example

Find 1110- 101



Answer: *1110 - 101 = 1001*

In the subtraction of two binary numbers there are only three different possible outcomes: (denary)

0 - 0 = 0 (0)
10 - 0 = 10 (2)
10 - 1 = 1 (1)
11 - 1 = 10 (2)

Exercise K

1 Work out the answer to each binary subtraction. Give your answer in binary.

a 100 - 10	b 101 - 10	c 111 - 100
d 1000 - 110	e 11101 - 10100	f 111010 - 101010
g 1010101 - 10010	h 1101011 – 1011000	i 1000 - 111
j 1101101 - 101100	k 1110110 – 1001010	l 11101011 - 1001010

2 Tom and Ravinder are playing a game. For each round whoever has the highest score wins a point if their answer, in denary (decimal), is lower.

Find the value for each player in each round.
Work out which player wins each round. (Convert their total to denary)
Write down who wins the game and their score.

	Tom	**Ravinder**
Round 1	1010 - 111	1010 - 1001
Round 2	101110 - 10011	100010 - 11010
Round 3	101100 - 100101	111011 - 101110
Round 4	111000 - 101101	111011 - 101010
Round 5	111101 - 110001	1011011 - 1000110

Multiplication using a Binary Shift

The multiplication operation is different using the binary number system.

A **binary shift** is used to move the binary digits which gives them a different place value.

In denary if you moved a number to the left it would become 10 times larger.

For example:

	X10	x10	x10		
Decimal (denary)	1000s	100s	10s	1s	
			← 8		8 x 10 = 80
			8	0	
		← 9	4		90 x 10 = 900
		9	**4**	**0**	4 x 10 = 40
					900 + 40 = 940

The **least significant** place is then filled with 0.

So 94 x 10 would become 940.

The same principle applies in binary.

A **left shift** is used to multiply in binary.

If a number is moved to the left it is multiplied in terms of the place value.

Each position makes the number 2 times larger or multiplies it by 2.

Example

	X2	*x2*	*x2*		
Binary	*Eight*	*Four*	*Two*	*Units*	
			← *1*		*1 x 2*
			1	***0***	*= 10 (2)*

	X2	*x2*	*x2*		
Binary	*Eight*	*Four*	*Two*	*Units*	
		← *1*		*0*	*10 x 2*
		1	***0***	***0***	*= 100 (4)*

As with denary, the least significant places (columns to the right) that are empty are then filled with a 0.

So 10 (2) would become 100 (4).

Example

	X2	*x2*	*x2*		
Binary	*Eight*	*Four*	*Two*	*Units*	
	←		*1*	*1*	*11 x 4*
	1	***1***	***0***	***0***	*= 1100 (12)*

A 12. 12 is 1100

The least significant place value is always filled with a 0 when multiplying binary.

The number has been **shifted to the left** or **a left shift** has occurred.

You can multiply by the following numbers:

- Moving 1 position to the left multiplies by 2
- Moving 2 positions to the left multiplies by 4
- Moving 3 position to the left multiplies by 8
- Moving 4 positions to the left multiplies by 16
- Etc.

Exercise L

1 Multiply the following binary numbers by the given number. Give your answers in binary

a 101 x 2	b 111 x 2	c 1011 x 2	d 1110 x 4
e 1111 x 2	f 10000 x 4	g 11101 x 2	h 11101 x 4
i 11110 x 2	j 100101 x 2	k 111010 x 4	l 111011 x 2

2 In each calculation:

i) Convert the given binary number to denary
ii) Multiply the binary number by the denary number and give your answer in binary and denary

a 10101 x 4	b 11101 x 4	c 110110 x 4	d 1101 x 8
e 11001 x 4	f 11011 x 4	g 10110 x 8	h 1101011 x 2
i 111011 x 4	j 1011 x 8	k 101101 x 4	l 1111 x 8

Division using a Binary Shift

The division operation is very similar to the multiplication operation when using the binary number system. A binary shift is used to move the binary digits to give them a different place value.

A **right shift** is used for division.

If a number is shifted to the right it has the opposite effect to that of a left shift.

The number is divided by 2 for each column that it is moved.

Example

	X2	*x2*	*x2*		
Binary	*Eight*	*Four*	*Two*	*Units*	
		1 →	*1*	*0*	*110 / 2*
			1	***1***	*= 11 (3)*

As the example above shows, in denary the starting number would be 6, and the calculation 6 divided by 2 equals 3. 3 is 11.

If the least significant places that are lost are zero then they are removed from the end. So 110 (6) would become 11 (3).

Example

	X2	*x2*	*x2*		
Binary	*Eight*	*Four*	*Two*	*Units*	
1 →		*1*	*1*	*0*	*1110 / 2*
		1	***1***	***1***	*= 111 (7)*

As the example above shows, in denary the starting number would be 14, and the calculation 14 divided by 2 equals 7. 7 is 111.

If the least significant place is a 1, then the outcome differs slightly. Shifting to the right gives integer division. The last digit is still lost in the shift, and you need to be aware that your answer is rounded down.

Example

	X2	*x2*	*x2*	
Binary	*Eight*	*Four*	*Two*	*Units*
1		*0*	*1*	*101 / 2 = 10 (2)*
→		***1***	***0***	

In denary this example would be 5 / 2 = 2.5

In binary the example is 101 / 2 = 10 (10 is 2 in denary)

This is still the correct answer, although you need to be aware that the remainder/decimal has been lost. This may mean that you answer is not exact. This method gives the **quotient**.

You can divide by the following numbers:

- Moving 1 position to the right divides by 2
- Moving 2 positions to the right divides by 4
- Moving 3 positions to the right divides by 6
- Moving 4 positions to the right divides by 8
- Etc.

Exercise M

1 Divide the following binary numbers by the given denary number.

a 110 / 2	b 100 / 2	c 1010 / 2	d 1100 / 4
e 1101 / 2	f 10000 / 4	g 11101 / 2	h 11100 / 4
i 11110 / 2	j 100101 / 2	k 110110 / 4	l 111101 / 2

2 For each of the following calculations:

i) Convert the binary number to denary
ii) Divide the binary number by the denary number and give your answer in both binary and denary

a 10100 / 4	b 11000 / 4	c 110110 / 4	d 110111 / 4
e 11101 / 4	f 110101 / 4	g 1011000 / 8	h 1101010 / 4
i 111011 / 8	j 1010011 / 8	k 10110100 / 8	l 11011100 / 8

Compression

Frequency

In maths data is sorted into tables to make it easier to understand and interpret.

Frequency tables are used to sort data to make it easier to read or interpret.

This is necessary when there is a set of data with a large number of different values.

This set of data does not need to be recorded in a frequency table:

Crystal records how many pieces of homework she is give on every school night for one week:

4 1 3 5 3

However, if Crystal recorded her homework for a term or a year she would have too much data to record. She would sort her data into a *frequency table.*

Crystal is never given more than 6 pieces of homework on one night. She keeps a record for a year, counts (tallies) her results and puts them in a *frequency table.*

Homework for one year:

Number of pieces in one school night	Frequency
0	21
1	42
2	52
3	61
4	12
5	6
6	1
Total	195

Tables are used to sort data in Computer Science.

Exercise A

1 This frequency table records the pieces of homework set each night for Stuart.

Number of pieces in one school night	Frequency
0	21
1	42
2	52
3	67
4	12
5	0
6	1
Total	195

a How often was Stuart given six pieces of homework in one night?

b What was the most frequent number of pieces of homework Stuart was given?

c How often was Stuart given two pieces of homework?

d Was Stuart set one piece of homework or four pieces of homework more often? Find the difference in the frequency for these values.

e How many days are there in total in Stuarts's school year?

f Was Stuart set homework on every night of the year? Explain your answer.

2 Jacob and Kayleigh are using two six-sided dice to find the probability that they will score 12 when the numbers of the dice are added together.

They each roll the dice 100 times.

Their results have been recorded in a frequency table.

These are Jacob's results:

Sum of two dice	Frequency
2	1
3	2
4	3
5	8
6	20
7	28
8	18
9	9
10	4
11	5
12	2
Total	100

These are Kayleigh's results

Kayleigh	
Sum of two dice	Frequency
2	3
3	0
4	4
5	6
6	17
7	25
8	11
9	11
10	10
11	8

12	5
Total	100

a Why does the table start with 2 and not 1?

b Which player rolled 12 most often?

c How many more times did Jacob roll a 7 than Kayleigh?

d Which player scored more than 10 most often? Show your calculations to explain your answer.

e Which player scored the most number of even scores? Show your calculations to explain your answer.

3 Four candidates stand for election as Head Student; Candidate E, Candidate F, Candidate G and Candidate H. The person with the most votes wins and the person who comes second will act as Deputy Head Student.

H	H	E	H	G	H	E	H	F	E	H	F	H
E	E	E	F	F	H	H	E	F	H	E	H	H
F	H	H	H	H	F	H	E	F	E			

a Draw a frequency table of these results. Include a column to tally the results.

b How many votes were recorded in total?

c Who won the election? How many votes did they get?

d Who should be Deputy Head Student? Explain your answer.

Types of Compression

In computing, **compression** is used to reduce the number of bits it takes to store a particular piece of data. This makes the size of the file used to store this data smaller.

This data could be a piece of text, an image, a piece of sound etc.

Compression can be split into two categories, **lossless** and **lossy**.

Lossless and lossy compression work very differently.

Lossless compression reduces the size of a file and allows the original file to be recreated exactly.

This means that none of the original information is lost and when the file is **decompressed/uncompressed**, the file will be EXACTLY as it was to begin with. The quality is NOT reduced.

Lossy compression removes or eliminates "unnecessary" pieces of information to make the file smaller.

For example, if a picture had a lot of blue sky, there would be different shades of blue. Instead of storing all the different shades, it would just store a few. Storing less information would 'compress' the file and make it smaller.

If pieces of information are removed, then the image will not be able to be put back to EXACTLY how it was to begin with, so the quality is reduced.

Compression of Data using Huffman Coding / Encoding

Huffman encoding is an algorithm used to reduce the size of a message that needs to be sent or stored. Huffman encoding reduces the number of bits required to store the message, which reduces its size.

Huffman encoding is a lossless method of compression.

Without using compression it takes 8 bits to store one character (assuming the characters are stored using ASCII)

It would take 136 bits (17 * 8 = 136) to store this message:

ANNA ATE AN APPLE

Using Huffman encoding, letters with a higher frequency are stored using a smaller number of bits. This reduces the number of bits and the space required to store a message.

If we store the most frequent letters using a smaller number of bits it reduces the number of bits overall e.g.

A appears most frequently so is stored using the least amount of bits.

N appears next most frequently so is stored using a slightly larger number of bits.

Etc.

L appears joint least frequently, is represented using the largest number of bits.

A	0
N	10
space	110
P	1110
E	11110
T	111110
L	111111

A	N	N	A		A	T	E		A	N	
1	2	2	1	3	2	6	5	3	1	2	3

A	P	P	L	E
1	4	4	6	5

Using this system would enable the same message to be stored using only 51 bits. This has reduced the number of bits by 136-51 = 85

Creating a Huffman Tree

A **Huffman tree** is drawn to show the different values and their frequencies.

This can then be used to compress the data.

Example

1 Draw a Huffman tree for the following set of data
2 Encode the letters
3 Find out how many bits have been saved by compressing the data using a Huffman Tree

A	B	A	B	C	B	A	B	D	D	A	D	D	C
C	B	A	D	E	D	D	D	D	D	B	B	C	D
C	C	D	D	C	C	D	D	D	D	C	D	D	C

Answer

a) Step One: Sort the data by frequency

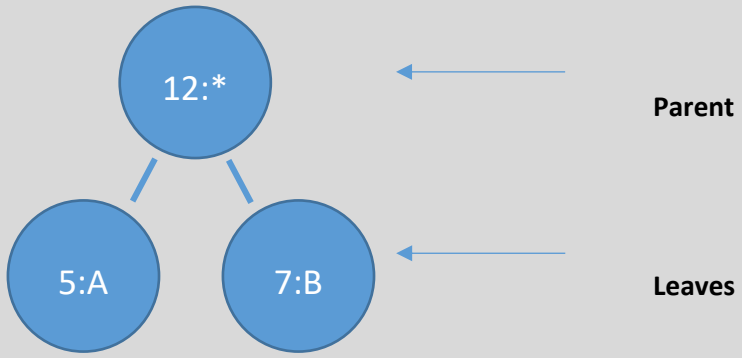
Value	**Frequency**
A	5
B	7
C	10
D	17

Step Two: Find the two values with the lowest frequencies. Add the frequencies.

	Value	**Frequency**
A		5
B		7
C		10
D		17

5 + 7 = 12.

Step Three: Draw the first three nodes of the Huffman Tree with 12 as the parent (shown by a *) and the two values as the leaves.



Step Four: Rewrite the table with the compressed data in place of the two lowest values.

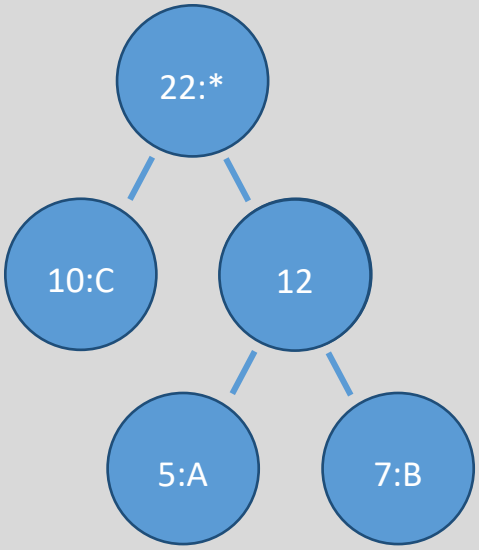
Value	**Frequency**
C	10
*	12
D	17

Step Five: Repeat the process; combine the two lowest values.

Value	Frequency
C	10
*	12
D	17

10 + 12 = 22,

Step Six: Create a new parent node of 22.



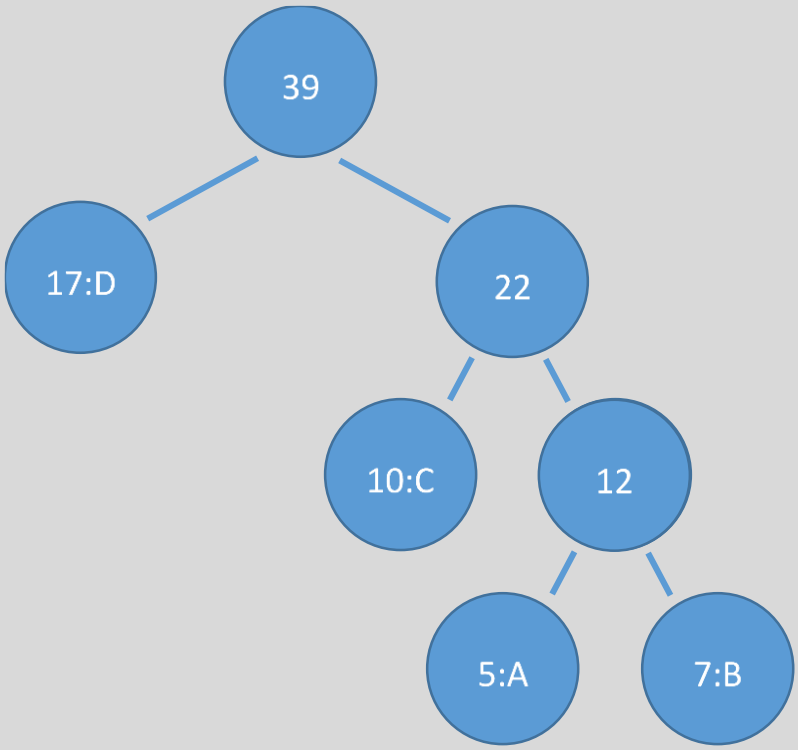
Step Seven: Rewrite the table with the compressed data in place of the two lowest values

Value	Frequency
D	17
*	22

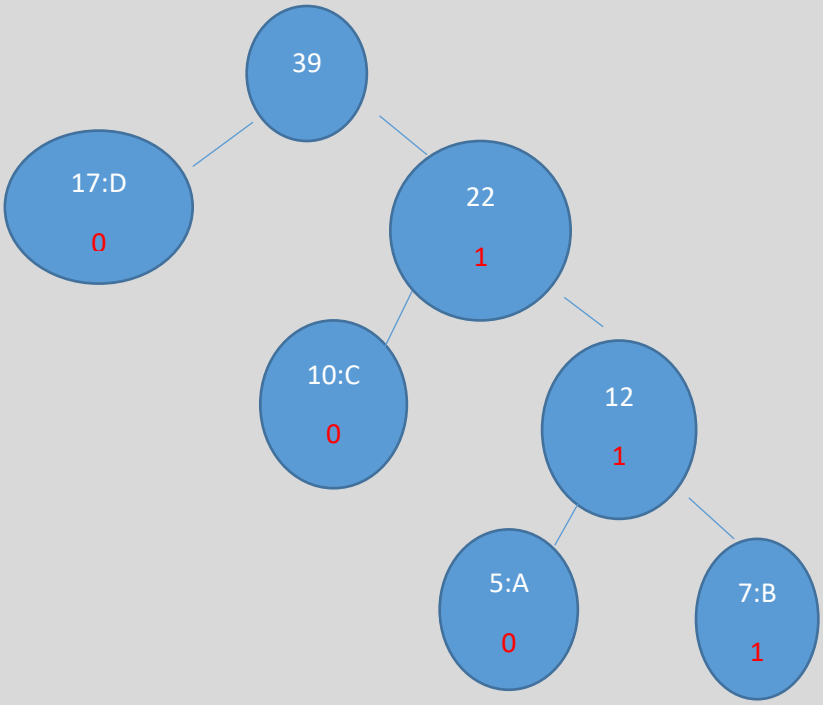
Step Eight: Repeat the process; combine the remaining two values

Value	Frequency
D	17
*	22
	17+22=39

Step Nine: Create a new parent node of 39.



All numbers have been included within the Huffman tree. The Huffman Tree is complete for this set of data.
Step Ten: Encode using 1 and 0 for each pair of leaves.



b) Reading the codes from the parent node downwards letters are coded as

D:0 C:10 B:111 A:110

c) Without compression the data uses 39*16 bits = 624 bits.

With compression using a Huffman Tree the data uses 5*3+7*3+10*2+17*1=15+21+20+17=73 bits

Saving = 624-73=551bits.

Exercise B

1 a) Draw a Huffman tree for the following set of data
b) Encode the letters
c) Find out how many bits have been saved by compressing the data using a Huffman Tree

A	B	B	C	D	A	B	A
B	C	D	B	C	B	B	B
B	B	A	B	A	C	B	A

2 a) Draw a Huffman tree for the following set of data
b) Encode the letters
c) Find out how many bits have been saved by compressing the data using a Huffman Tree

D	B	B	B	D	B	B
B	C	D	B	C	B	B
B	B	A	D	B	B	B
A	B	B	C	C	D	D
D	A	D	A	B	C	B

3 a) Draw a Huffman tree for the following set of data
b) Encode the letters
c) Find out how many bits have been saved by compressing the data using a Huffman Tree

E	D	B	C	E	E	A	E	B	D
B	E	B	E	A	E	E	B	B	B
A	B	E	E	E	B	B	A	C	C
B	B	E	E	A	E	A	C	E	E

4 a) Draw a Huffman tree for the following set of data
d) Encode the letters
e) Find out how many bits have been saved by compressing the data using a Huffman Tree

Letter	**Frequency**
A	8
B	13
C	7
D	5
E	15
F	9

Compression of Data using Run Length Encoding

Run-length encoding is an algorithm used to reduce size of a file/piece of data. Run length encoding reduces the number of bits required to store the file by combining repeating pieces of data. This data is called a **run** and is typically encoded into two **bytes**.

Example

The piece of text below contains repetition, the letter A is repeated 15 times:

AAAAAAAAAAAAAAA

It would usually take 15 bytes to store this information. 1 byte = 1 character.

The same string of characters can be stored as **15A**.

The **run count**. The amount of times that the letter or number is repeated.

The **run value**. The value which has been repeated.

As stated above, this would then be stored as two bytes, not 15. A byte to store the run count and a byte to store the value.

Example

The piece of text below contains repetition, although more than one letter is repeated.

AAAAAAbbbXXXXXt

This would be grouped as:

6A
3b
5X
1t

This would be displayed together as:

6A3b5X1t

Numbers can also be compressed using the same method.

Example

The number below contains repetition, although more than one digit is repeated.

0000011100000011

This would be grouped as:

50
31
60
21

Exercise C

1 Compress the following data using Run-length encoding.

AAAbbHHHHHHHHZZ

2 Compress the following data using Run-length encoding.

11110000111111100000000111

3 Compress the following data using Run-length encoding.

LLLLLLLkkkkkkkkkKKKKKKSSSSSAAAAAAADDDDDD

4 Compress the following data using Run-length encoding.

aaaaAAAAGGGGffffffffHHHHHHHHHHZZZZZAAAAAAAa

5 Compress the following data using Run-length encoding.

11100111101111100000001001110

Algorithms

An **algorithm** is a sequence of steps that can be followed to complete a task or solve a problem.

You will follow algorithms on a daily basis to complete many different tasks.

Examples of algorithms could be things like following a food recipe, making a cup of tea, following a set of directions or the instructions required to complete a particular problem in mathematics.

Algorithms in Mathematics

You will be familiar with **rules** and **methods** in maths.

When you follow a rule or use a method solve an equation you are using an algorithm because you are solving a problem and trying to find a solution.

Example

To answer this question you will use the balancing method to solve the equation.

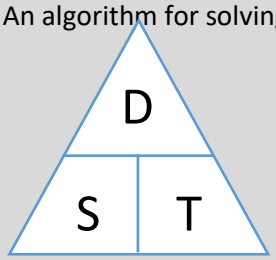
Solve	3y + 4 = 19	{subtract 4 from both sides}
	3y= 15	{divide both sides by 3}
	y = 5	{the solution}

The sequence of steps can be represented as a set of **operations**:

For this equation this sequence of steps will always find the correct answer/solution so this is an algorithm.

However, this method is **not** an algorithm for solving **all** equations – it only works for this equation.

An algorithm for solving **any** equation would need to work for every possible equation.



Distance = Speed x Time

'To calculate distance multiply speed by the time taken'.

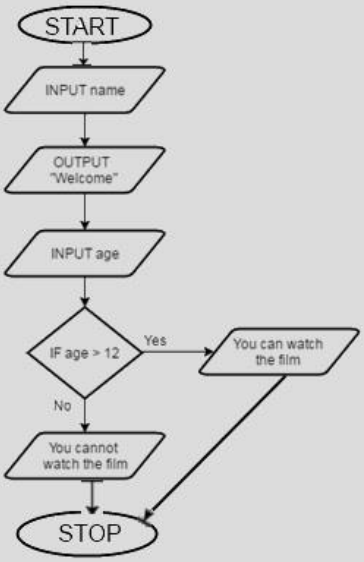
This is always true. $\boldsymbol{D = S \times T}$ is an algorithm for finding distance.

Algorithms in Computer Science

Pseudocode Example

```
BEGIN
INPUT name
OUTPUT "Welcome"
INPUT age
IF age > 12 THEN
      OUTPUT "You can watch the film"
ELSE
      OUTPUT "You cannot watch the film"
ENDIF
END
```

Flow Diagram Example



Variables and Constants

In mathematics when a letter is used in algebra it is representing a **variable**.

Once the value of the variable is known it can be **substituted** for the letter.

Example

Find the value of this **expression** when b = 4, f= 5 and p = -3

$$b + f + p$$

Answer

$$b + f + p$$
$$= 4 + 5 + -3$$
$$= 9 - 3$$
$$= 6$$

A **constant** is a value that does not change. In this term 4 is a constant and m is a variable:

$$4m$$

In Computer Science, variables and constants are used within algorithms to store data.
This data can be a piece of text, an integer, a date etc.

Variables and constants often look the same, but it is the data which they hold that is important.
Within an algorithm the data held by a variable may change somewhere during the algorithm.
This change could be that it is overwritten completely, or altered slightly.
In contrast, a constant holds a value which will stay the same throughout the algorithm.

Example
In the algorithm below, the data contained within the total and subtotal variables changes whereas the data contained within the vat constant does not.

```
VAT =  1.2
subtotal = 0
total = 0
subtotal = 2.00 + 1.00 + 2.00
total = subtotal * vat

OUTPUT total
```

Creates a constant called vat with the value 1.2

Creates a variable called subtotal with the value 0

Creates a variable called total with the value 0

Adds 2.00, 1.00 and 2.00 and stores it in subtotal

Multiplies subtotal (5.00) by vat (1.2) and stores in total

Outputs the contents of the total variable

Variable Declaration

The **declaration** of a variable is the first time that a variable is created.
This creates a container to enable us to store a value later.

Declarations can look different depending on the programming language that you are using or the method you are using to design an algorithm e.g. pseudocode, flow diagram etc.

The example below declares a variable and assigns it a name.

Example

```
var variableName

var number1
```

Depending on how you are designing or programming your algorithm, it may be that you have to state the specific type of variable that you are creating e.g. is it going to store text? Is it going to store a number? Etc.

Example

```
int variableName;

char variableName
```

NOTE: 'int' would store an integer and 'char' would store characters/text.

In python, a variable is not declared until it is assigned a value, this is covered in the next section.

Example (python)

```
variableName = 10
variableName = "Mr Smith"
```

NOTE: Text is usually enclosed in speech marks to show that text is being used.
*Text in an algorithm is known as a text **string**.*

Numbers are not enclosed in speech marks to show that a number is being used.
*Whole numbers in an algorithm are known as **integers**.*

Assignment

As previously covered, variables and constants are used to store data values. In an algorithm, putting this value into a variable or constant is known as **assignment**.

A single equal sign (=) is used to show that a value is being assigned to a variable or constant. This explains why the relational operator for 'equal to' is a double equal sign (==).

Example

```
variableName = 15
variableName = "Hello World"

age = 45
name = "Mr Smith"
```

After a value has been assigned to a variable it can then just be referred to by the name that it has been given.

Example

```
name = "Mr Smith"
OUTPUT name
```

The above algorithm would store the words Mr Smith in the variable called name and then output the contents of name.

Output

Mr Smith

Assignment can occur multiple times in an algorithm. If the same variable name is used, the data is overwritten. Sometimes this is something which you want to happen.

Example

```
wage = 10
wage = wage * 4
OUTPUT wage
```

The above algorithm uses **substitution**. The algorithm would store the number 10 in the variable called wage. It would then substitute its original value with itself multiplied by 4. This would mean that value would then equal 40. It then outputs the contents of wage.

Output

40

Exercise A

1 Using the algorithm below, work out the value of the variable 'number3' using the variable values given.

```
number1 = _________
number2 = _________

number1 = number1 * 2
number2 = number2 + 18
number3 = number1 + number2

OUTPUT number3
```

a number1 = 10, number2 = 20

b number1 = 15, number2 = 9

c number1 = 12, number2 = 16

d number1 = 40, number2 = 25

e number1 = 19, number2 = 33

f number1 = 27, number2 = 13

2 Using the algorithm below, work out the value of the variable 'number3' using the variable values given.

```
number1 = _________
number2 = 5

number1 = number1 * 5
number1 = number1 + number2
number3 = number1 - 14

OUTPUT number3
```

a number1 = 10

b number1 = 8

c number1 = 15

d number1 = 27

e number1 = 33

f number1 = 45

Selection

In algorithms, **selection** is when a question is asked, and depending on the answer, the program follows a given course of action. Depending on the answer to this question a different path through the algorithm may be taken. Selection statements are also known as **if statements** or **conditional statements**.

Selection could be used to give a different course of action depending on the answer to a question in the real world.

Example

If you are male then it is recommended to eat 2500 calories, but if you are female it is recommended to eat 2000 calories.

In an algorithm, this might be written as:

```
IF gender == "male" THEN
      calories = 2500
ELSE
      calories = 2000
ENDIF
```

This selection statement only asks a single question - if they are male. In this situation there is only one other possible answer. The 'else' statement means that anything else other than male will result in the calories equalling 2000.

So if `gender = "female"` the answer would be false and calories would equal 2000.

NOTE: The if, else and endif are in line with each other. This shows that they are connected. The indented instruction is only selected for the part of the question which is yes/true.

If we are asking a question with multiple answers, then the statement will differ slightly.

Example

```
number = 122

IF number > 100 THEN
      OUTPUT "Large Number!"
ELSEIF number > 50 THEN
      OUTPUT "Medium Number!"
ELSE
      OUTPUT "Small Number!"
ENDIF
```

There can be an unlimited number of 'elseif' statements added. The 'else' at the end of the statement would mean that if any of the earlier questions are not yes/true, then that will be returned.

E.g. if 30 were entered, none of the earlier questions would be yes/true, so "Small Number!" would be returned.

Exercise B

1 Using the algorithm below, work out whether the algorithm will output "older than you" or "not older than you" using the variable values given.

```
dog_age = _________________
dog_years = dog_age * 7
person_age = _________________
IF dog_years > person_age THEN
    OUTPUT "Your dog is older than you!"
ELSE
    OUTPUT "Your dog is not older than you."
```

a person_age = 10 dog_age = 3

b person_age = 19 dog_ age = 6

c person_age = 47 dog_ age = 8

d person_age = 53 dog_ age = 9

e person_age = 42 dog_ age = 7

f person_age = 49 dog_ age = 7

2 Using the algorithm below, work out what will be output when number is output at the end of the algorithm. Use the variable values given.

```
number = _________
IF number >= 64 THEN
    number = number / 3
ELSEIF number >= 32 THEN
    number = number * 4
ELSEIF number >= 16 THEN
    number = number -  23
ELSEIF number >= 8 THEN
    number = number * 8
ELSE
    number = number + 9
OUTPUT number
```

a number = 66

b number = 31

c number = 32

d number = 14

e number = 30

f number = 70

g number = 76 h number = 7 i number = 84

Nested selection

Selection statements can be included inside each other to ask another question based on the answer to the first question e.g. Are you in year 10? Are you in tutor group 10.3? Etc. This is known as **nested selection**.

Example

Once the answer to the first question has been determined e.g. is the person's age less than 12, depending on the answer the second question may be asked.

```
age = 14
height = 1.1
IF age < 12 THEN
      OUTPUT "Sorry, you are not old enough to go on this ride"
ELSE
      IF height <= 1.2 THEN
            OUTPUT "Sorry, you do not meet the height requirements"
      ELSE
            OUTPUT "Join the queue!"
      ENDIF
ENDIF
```

The above algorithm would answer false for the first selection statement as the individual is 14, so would move to the instructions contained within the else part of the statement. This would then cause the second selection statement to be carried out and check whether the person less than or equal to 1.2m. As the person is 1.1m the second selection statement would be true and the output be "join the queue!".

Exercise C

1 Using the algorithm below, work out the cost of the postage for each parcel using the variable values given.

```
country = _________
item1 = _________
item2 = _________
item3 = _________
```

```
total = item1 + item2 + item3

IF country == "US"
      IF total <= 50
            OUTPUT "Shipping Cost is £14.99"
      ELSEIF total <= 100
            OUTPUT "Shipping Cost is £9.99"
      ELSEIF total <= 150
            OUTPUT "Shipping Costs £4.99"
      ELSE
            OUTPUT "FREE"
      ENDIF
ELSEIF country == "UK":
      IF total <= 50:
            OUTPUT "Shipping Cost is £4.99"
      ELSE
            OUTPUT "FREE"
      ENDIF
ENDIF
```

a item1 = £47 item2 = £29 item3 = £18 country = UK

b item1 = £19 item2 = £24 item3 = £21 country = UK

c item1 = £14 item2 = £21 item3 = £9 country = US

d item1 = £51 item2 = £37 item3 = £20 country = US

e item1 = £37 item2 = £64 item3 = £9 country = UK

f item1 = £21 item2 = £33 item3 = £16 country = UK

g item1 = £11 item2 = £26 item3 = £31 country = US

h item1 = £6 item2 = £19 item3 = £47 country = UK

2 Using the algorithm below, work out which grade each student will receive using the variable values given.

```
result = _____________
IF result >= 90 THEN
     IF result >= 97 THEN
          grade = "A*"
     ELSEIF result >= 95 THEN
          grade = "A+"
     ELSEIF result >= 93 THEN
          grade = "A="
     ELSE
          grade = "A-"
     ENDIF
ELSEIF result >= 80 THEN
     IF result >= 86 THEN
          grade = "B+"
     ELSEIF result >= 83 THEN
          grade = "B="
     ELSE
          grade = "B-"
     ENDIF
ELSEIF result >= 70 THEN
     IF result >= 76 THEN
          grade = "C+"
     ELSEIF result >= 73 THEN
          grade = "C="
     ELSE
          grade = "C-"
     ENDIF
ELSE
     print "You are required to retake the test!"
```

```
ENDIF
```

a result = 94 b result = 88 c result = 80 d result = 72

e result = 87 f result = 70 g result = 76 h result = 68

i result = 91 j result = 82 k result = 95 l result = 99

Definite Iteration

In an algorithm, **iteration** is the act of repeating part or all of the instructions. This is also referred to as a **loop**. Definite iteration repeats instructions a fixed amount of times.

Example

In definite iteration, a variable is used to record how many iterations have taken place, or how many times the instructions are repeated.

The iteration starts with the first number and stops when the second number has been reached. It is called definite iteration because the code will only repeat this set number of times.

```
FOR variableName = 0 to 5 DO
      OUTPUT "Hello"
NEXT variableName
```

Controls how many times the code repeats

The instruction which is repeated

Instructs the algorithm to repeat again.

The first time the algorithm runs, the variable is given the first number, in this case 0.

When the NEXT instruction is reached, the variable is changed to the next number, in this case 1. The instructions are repeated again.

When the second number is reached the iteration is stopped.

This means that the instructions are repeated when the variable is 0, 1, 2, 3, 4 and 5. When the variable reaches 5 the code stops. The output for this algorithm is shown below.

Variable Value	Output
variableName = 0	Hello
variableName = 1	Hello
variableName = 2	Hello
variableName = 3	Hello
variableName = 4	Hello
variableName = 5	Hello
variableName = 6	Iteration stops, nothing is output

Example

```
number = 0
FOR count = 0 to 4 DO
      number = number + 1
      OUTPUT number
NEXT count
```

The above algorithm will add 1 to the number variable each time it is run. It will then output the value of number before repeating the sequence. The algorithm will generate the output below.

Variable Value	Output
count = 0	1
count = 1	2
count = 2	3
count = 3	4
count = 4	5
count = 5	Iteration stops, nothing is output

Exercise D

1 Work out the output for the algorithm below.

```
number = 6
FOR count = 0 to 15 DO
      number = number * 2
      number = number - 4
      OUTPUT number
NEXT count
```

a Work out the output generated by the first 8 iterations for the algorithm above. A similar table to the one in the example above can be used.

b How many times will the above algorithm iterate?

c If the value of the number variable is changed to 10, work out the output generated by the first 8 iterations for the algorithm above.

2 Using the algorithm below, work out what will be output on the given iterations.

```
number = 4
FOR count = 3 to 13 DO
      number = number * 2
      IF number < 25 THEN
            OUTPUT "True"
      ELSE
            OUTPUT "False"
NEXT count
```

a Work out the output generated by the first 8 iterations for the algorithm above. A similar table to the one above can be used.

b How many times will the above algorithm iterate?

c If the value of the number variable is changed to 10, work out the output generated by the first 8 iterations for the algorithm above.

Indefinite Iteration

Indefinite iteration is also referred to as a **loop**. Indefinite iteration repeats instructions until a **condition** is met. This means that this type of iteration will not always repeat instructions the same amount of times.

This type of iteration is used when it is not known exactly how many times the algorithm will loop.

```
WHILE answer != "computer" DO
      answer = INPUT "Input password"
ENDWHILE
```

While the condition is false the loop will continue

The instructions which are repeated

Causes the loop to end when the condition is not true been met

*Note: **INPUT** "Input password" would output a message saying "Input password" and allow a password to be entered.*

The above algorithm checks a password that has been entered. If the password is incorrect (answer != "computer") then the loop continues to iterate.

As it is an indefinite loop it will continue to iterate while the condition is true. When the condition is not true, in this case if answer == computer, the iterations will stop.

For example, if the correct password was entered incorrectly twice and then entered correctly, the output would be as below.

Output	Input	Loop Condition
Input password	answer = "comput"	"comput" != "computer"
Input password	answer = "compute"	"compute" != "computer"
Input password	answer = "computer"	"computer" = "computer"
Iteration stops, nothing is output.		

In an indefinite iteration, the variable used within the condition can be changed by the algorithm in order to control how many iterations of the loop occur.

```
count = 0
WHILE count < 50 DO
      count = count + 5
ENDWHILE
```

The count variable is initially set to 1

The iteration will continue while count is less than 50.

Increases the count variable by 5 each time the statement iterates – 1st iteration: count = 5, 2nd iteration: count = 10, 3rd iteration count = 15. This continues until count is >= 50.

Exercise E

1 Using the algorithm below, work out the iterations for each of the different variable values.

```
total = 1
WHILE total < 20 DO
    OUTPUT total
    total = total + 1
ENDWHILE
```

a Work out the output generated by the first 8 iterations for the algorithm above. A similar table to the one above can be used.

b How many times will the above algorithm iterate?

c If the value of the variable `total` is changed to 10, how many times with the above algorithm iterate?

2 Using the algorithm below, work out how many times the statement would iterate for each of the different variable values.

```
total = _________
WHILE total < 100 DO
    OUTPUT total
    total = total * 2
ENDWHILE
```

a total = 12 b total = 18 c total = 7

d total = 11 e total = 14 f total = 2

Subroutines

In algorithms, **subroutines** are often used to reduce the number of instructions when there is a particular task or part of the code which must be repeated.

Rather than repeating the same instructions multiple times, these instructions are written once and then used elsewhere in the code.

Using the same instructions over and over again, without duplicating them, means that the number of instructions within the algorithm are reduced. This is to make algorithms more **efficient**.

Efficiency is very useful if they are later turned into computer programs.

Creating a subroutine:

```
SUBROUTINE
identifier(parameters)
      # instructions here
ENDSUBROUTINE
```

- Identifies the beginning of the subroutine
- The name of the subroutine and any parameters
- The instructions which will be reused
- Identifies the end of the subroutine

Example

Below is a simple subroutine that stores the word 'Hi' into a variable called 'welcome'.

```
SUBROUTINE
say_hi()
      welcome = "Hi"
ENDSUBROUTINE
```

The subroutine is given a name, which in this example is 'say_hi'. This allows other parts of the algorithm to refer to this sequence of instructions.

'**Calling**' a subroutine means that the sequence of instructions inside the subroutine are **executed**.

This can be done multiple times within the same algorithm if needed.

A subroutine is called by using its name. The name of this subroutine is 'say_hi'.

You can assign the output from the subroutine to a variable.

Example

```
greeting = say_hi()
```

You can the output the result of the subroutine.

Example

```
OUTPUT say_hi()
```

Instead of doing something as simple as displaying a word, subroutines can be used to do more complicated things such as perform regularly used calculations.

This is very similar to how some algebra works.

Example

The algorithm below is set out in the same way as the first example. It is given the name 'my sum'.

The variable names within the brackets are called **parameters**. These parameters are used later in the subroutine to store the numbers which we want to add up.

```
SUBROUTINE
my_sum(a, b)
      result = a + b
      RETURN result
ENDSUBROUTINE
```

Once a subroutine has been created it can be called later in the algorithm as many times as it is needed.

```
answer = my_sum (2, 3)
OUTPUT my_sum (2, 3)
```

The subroutine adds a and b and stores them in a variable called result. This variable is then output.

The numbers within this subroutine may change each time it is executed, which is why we use parameters/variables and not the numbers themselves.

Whichever numbers are placed in the brackets will be used within the sum – like in algebra, the a and b represent these numbers.

The 2 will be stored in parameter a and the 3 will be stored in the parameter b.

Whichever numbers are contained within the brackets are used within the subroutine.

Exercise F

1 Work out the output for the algorithm below, using the given parameters when the subroutine is called.

```
SUBROUTINE
triple(number)
      number = number * 3
      RETURN number
ENDSUBROUTINE

result = triple(__)
OUTPUT result
```

a triple(3) b triple(12) c triple(9)

d triple(15) e triple(31) f triple(43)

g triple(39) h triple(27) i triple(47)

2 Work out the output for the algorithm below, using the given parameters when the subroutine is called.

```
SUBROUTINE
bigger(number1, number2)
      IF number1 > number2 THEN
            answer = number1 * 4
      ELSE
            answer = number2 * 4
      ENDIF
      RETURN answer
ENDSUBROUTINE

result = bigger(__, __)

IF result >= 50 THEN
      OUTPUT "True"
ELSE
      OUTPUT "FALSE"
ENDIF
```

a bigger(6, 8)

b bigger(8, 7)

c bigger(9, 7)

d bigger(11, 16)

e bigger(13, 18)

f bigger(13, 17)

g bigger(21, 5)

h bigger(14, 20)

i bigger(17, 19)

Common Algorithms

Algorithms that Search: Binary Search

A binary search is an efficient algorithm for finding an item from an ordered list of items. e.g. a binary search could be used to find the number 8 in the following list of numbers 11, 12, 5, 13, 8, 1, 4, 7

To work the numbers have to be in order: 1, 4, 5, 7, 8, 11, 12, 13.

There are lots of different search algorithms e.g. linear search, comparison search, digital search etc. Different search algorithms search with different levels of efficiency. This means different algorithms may take more or fewer operations to find a particular piece of data within a list.

The ideal search algorithm will find the value you are looking for using the fewest possible operations.

Example

Firstly you must put the numbers into order from smallest to largest. In this example we are searching for the number **9**.

1	2	3	4	5	6	7	8	9	10
2	6	9	12	16	18	20	23	45	99

Find the middle value. In this example there are 10 values, so 10 / 2 = 5 - the 5th value.

1	2	3	4	5	6	7	8	9	10
2	6	9	12	16	18	20	23	45	99

Is this equal to, greater than, or smaller than 9? (The number which you are looking for)

16 is greater than 9 (16 > 9) so you know your number must be to the left of 16. Make a new list with all the numbers to the left.

1	2	3	4	5	6	7	8	9	10
2	6	9	12	16	18	20	23	45	99

We now need to search the new list for the number **9**.

1	2	3	4
2	6	9	12

Find the middle value again. There are now 4 values left. 4 / 2 = 2 so we are starting with the 2nd value.

1	2	3	4
2	6	9	12

Is this equal to, greater than, or smaller than 9?

6 is smaller than 9 (6 < 9) so we can remove the numbers to the left as we know our number is not there.

1	2	3	4
2	6	9	12

We can again search the new list for the number **9**.

3	4
9	12

Again, find the middle value. With only two numbers left 2 / 2 = 1 so we look at the 1st value.

3	4
9	12

Is this equal to, greater than, or smaller than **9**?

The 3rd number is equal to 9 so we have found our number!!

The binary search algorithm could be represented using the following set of instructions:

WHILE the number has not been found DO

GET the middle number

IF it is the value you are looking for.

Well done, your found your number!

ELSEIF it is larger than the one you are looking for

Take the values to the left of the middle value

ELSE

Take the values to the right of the middle value

ENDIF

REPEAT the instructions

Exercise A

1 Given the list 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, how many times would the algorithm have to repeat to find the value 30? List each of the steps that you take.

2 Given the list 17, 21, 33, 38, 55, 60, 72, 88, 94, how many times would the algorithm have to repeat to find the value 72? List each of the steps that you take.

3 Given the list 11, 40, 30, 48, 23, 19, 51, 44, 26, 29, how many times would the algorithm have to repeat to find the value 19? List each of the steps that you take.

Algorithms that Sort: Bubble Sort

A bubble sort is an algorithm for sorting data into a specific order, usually in order of size. E.g. a bubble sort could be used to sort 5, 19, 12, 8, 11, 16 into order from largest to smallest – 5, 8, 11, 12, 16, 19.

There are many different sorting algorithms e.g. insertion sort, selection sort, merge sort etc.

Different sorting algorithms sort data with different levels off efficiency. This means different algorithms may take more or fewer operations to sort data into the required order.

The ideal sorting algorithm will sort the data into the order that is required with the fewest possible operations.

A bubble sort works by comparing a piece of data to the next piece of data in a list. The data is then moved based on which value is bigger / smaller.

Example

The list of number that we begin with. The numbers are not in order. We want to sort the numbers from smallest to largest.

Element Number	1	2	3	4	5
Value	13	2	6	4	9

1) Compare elements 1 and 2

1	2	3	4	5
13	2	6	4	9

2) Is element 1 > element 2?
3) **Yes:** so swap them

1	2	3	4	5
2	13	6	4	9

4) Compare elements 2 and 3

1	2	3	4	5
2	13	6	4	9

5) Is element 2 > element 3?
6) **Yes**: so swap them

1	2	3	4	5
2	6	13	4	9

7) Compare elements 3 and 4

1	2	3	4	5
2	6	13	4	9

8) Is element 3 > element 4?
9) **Yes**: so swap them

1	2	3	4	5
2	6	4	13	9

10) Compare elements 4 and 5

1	2	3	4	5
2	6	4	13	9

11) Is element 4 > element 5?
12) **Yes**: so swap them

1	2	3	4	5
2	6	4	9	13

You have come to the end of the list. At the end of the <u>first pass</u> through the numbers the list has been changed from this:

1	2	3	4	5
13	2	6	4	9

To this:

1	2	3	4	5
2	6	4	9	13

It is still not in order, but it is getting closer!

If any changes have been made you must start again. This is to check that the numbers are now in order. If they are not further changes will then be made.

Note: Each time you go through the numbers and come to the end of the list it is known as a 'pass'.

1) Compare elements 1 and 2

1	2	3	4	5
2	6	4	9	13

Swap? **No**

1	2	3	4	5
2	6	4	9	13

2) Compare elements 2 and 3

1	2	3	4	5
2	6	4	9	13

3) Is element 2 > element 3?
4) **Yes**: so swap them

1	2	3	4	5
2	4	6	9	13

5) Compare elements 3 and 4

1	2	3	4	5
2	4	6	9	13

Swap? **No**

1	2	3	4	5
2	4	6	9	13

6) Compare elements 4 and 5

1	2	3	4	5
2	4	6	9	13

Swap? **No**

1	2	3	4	5
2	4	6	9	13

After the second pass through the numbers we are left with the following order:

1	2	3	4	5
2	4	6	9	13

You will notice that the numbers are now in the correct order!

Although this is the case, if any changes have been made you must start again! This is to check that the numbers are in order. If they are then no further changes will then be made.

Exercise B

1 Sort the list [7, 3, 5, 2] into order using the bubble sort algorithm. List the order of the numbers at the end of each pass.

2 Sort the list [3, 2, 5, 7, 4] into order using the bubble sort algorithm. List the order of the numbers at the end of each pass.

3 Sort the list [11, 3, 9, 1, 8, 7] into order using the bubble sort algorithm. List the order of the numbers at the end of each pass.

4 List [8, 2, 8, 7, 3, 1, 2] is being sorted using bubble sort. Fill in the blanks to show the list after each pass.

After the 1st pass: [2, 8, 7, 3, 1, 2, 8]

After the 2nd pass: [2, 7, 3, 1, 2, 8, 8]

After the 3rd pass: [2, 3, 1, 2, 7, 8, 8]

After the 4th pass:

After the 5th pass:

After the 6th pass:

5 List [1, 5, 8, 7, 6, 1, 7] is being sorted using bubble sort. Fill in the blanks to show the list after each pass.

After the 1st pass: [1, 5, 7, 6, 1, 7, 8]

After the 2nd pass: [1, 5, 6, 1, 7, 7, 8]

After the 3rd pass:

After the 4th pass:

After the 5th pass:

After the 6th pass:

After the 7th pass:

6 List [6, 8, 2, 1, 1, 9, 4] is being sorted using bubble sort. Fill in the blanks to show the list after each pass.

After the 1st pass: [6, 2, 1, 1, 8, 4, 9]

After the 2nd pass:

After the 3rd pass:

After the 4th pass:

After the 5th pass:

After the 6th pass:

After the 7th pass:

Completing a Trace Table for an Algorithm

A **trace table** is used to track the values of variables as they change throughout an algorithm.

Example

```
total = 0
total = total + 2 + 2
OUPUT total
```

Throughout the program, the total will have changed from 0 to 4.

This can be very useful when you have an algorithm and you are not getting the answer you expect when you have worked through all of the instructions.

This is also known as a **dry run** and would form part of your testing of an algorithm.

Example

```
1     greeting = "Hello"
2     FOR count = 0 to 5 DO
3           OUTPUT greeting
4     NEXT count
```

Note: Each instruction/line can be given a number so that they are easier to follow

This trace table is generated for the algorithm above.

	greeting	count	OUTPUT
1	Hello		
2		0	
3			Hello
2		1	
3			Hello
2		2	
3			Hello
2		3	
3			Hello
2		4	
3			Hello
2		5	
3			Hello

The columns are labelled with the variable / constant names and outputs.

Work through the algorithm one instruction at time.

Each time the variable/constant appears, make a note of the line number and its value.

Make a note of any outputs.

Continue this until the algorithm is completed.

Working through the algorithm one instruction at a time:

Line 1: The greeting constant is set to “Hello”.

Line 2: As it is a definite loop, the count variable is set to 0.

Line 3: Line 3 outputs the greeting constant which is “Hello”.

Line 2: The next iteration takes place. The count variable is set to 1.

Line 3: Line 3 outputs the greeting constant which is “Hello”.

Line 2: The next iteration takes place. The count variable is set to 2.

Line 3: Line 3 outputs the greeting constant which is “Hello”.

The iterations will take place until the count variable is set to 5. This will then cause the iteration/loop to stop having executed the 5th iteration.

Example

```
1     x = 1
2     WHILE x < 30 DO
3           x = x * 2
4           OUTPUT x
5     ENDWHILE
6     OUTPUT "Finished!!"
```

Note: Each instruction/line can be given a number so that they are easier to follow

The trace table for the algorithm above is on the next page.

	x	OUTPUT
1	1	
3	2	
4		2
3	4	
4		4
3	8	
4		8
3	16	
4		16
3	32	
4		32
6		Finished!!

Working through the algorithm one instruction at a time:

Line 1: The x variable is set to 1.
Line 2: As it is an indefinite loop, the x must be < 30 (x = 1).
Line 3: The x variable is multiplied by 2 and stored in (x = 2).
Line 4: Outputs the x variable, which is 2.
Line 2: As it is an indefinite loop, the x must be < 30 (x = 2).
Line 3: The x variable is multiplied by 2 and stored in (x = 4).
Line 4: Outputs the x variable, which is 4.
Line 2: As it is an indefinite loop, the x must be < 30 (x = 4).
Line 3: The x variable is multiplied by 2 and stored in x (x = 8).
Line 4: Outputs the x variable, which is 8.
Line 2: As it is an indefinite loop, the x must be < 30 (x = 8).
Line 3: The x variable is multiplied by 2 and stored in x (x = 16).
Line 4: Outputs the x variable, which is 16.
Line 2: As it is an indefinite loop, the x must be < 30 (x = 16).
Line 3: The x variable is multiplied by 2 and stored in x (x = 32).
Line 4: Outputs the x variable, which is 32.
Line 2: As it is an indefinite loop, the x must be < 30 (x = 32).
Line 5: As x is greater than 30 (x = 30) the iteration/loop is ended.
Line 4: Outputs Finished!!

Exercise C

1 Create and complete a trace table for the algorithm below. Use the table which has been drawn for you.

```
1    number = 10
2    FOR count = 1 TO 2 DO
3        number = number + 6
4        number = number - 4
5        OUTPUT number
6        OUTPUT "Keep looping!"
7    NEXT count
8    OUTPUT "Finished looping!!"
```

	number	count	OUTPUT

2 Create and complete a trace table for the algorithm below.

```
1    x = 6
2    FOR count = 1 TO 3 DO
3        answer = count * x
4        OUTPUT answer
5    NEXT count
6    OUTPUT "Finished!!"
```

3 Create and complete a trace table for the algorithm below.

```
1    number = 64
2    count = 1
3    WHILE number >= count DO
4        number = number / 2
5        count = count + 1
6    ENDWHILE
7    OUTPUT "Finished!!"
```

4 Create and complete a trace table for the algorithm below.

```
1    x = 60
2    total = 1
3    FOR count = 1 TO 6 DO
4        IF total < x THEN
5            total = total * 3
6            OUTPUT "Smaller"
7        ELSE
8            OUTPUT "Bigger!"
9    NEXT count
10   OUTPUT "Finished!!"
```

Logic Circuits

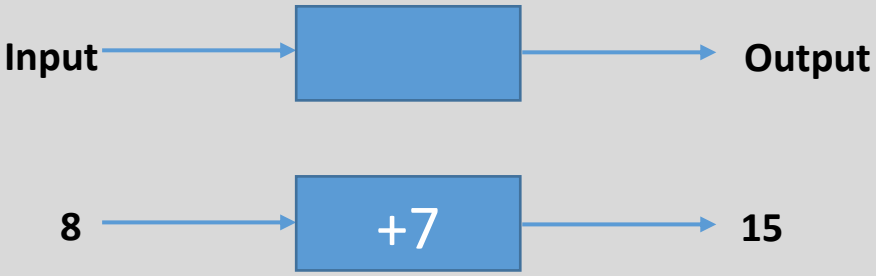
Logic Gates

In a computer, **logic gates** are used to make decisions based on the data which they receive. This data is in the form of electronic signals, which can be 1 'on' or 0 'off'. When using logic gates 1 is also referred to as 'true' and 0 as 'false'.

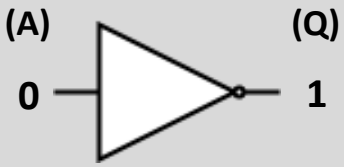
Individually logic gates can only make very simple decisions, but when used together they can make very complex decisions – they are the building blocks for what takes place in the **CPU**.

The symbols below work in the same way as number machines in mathematics.

An output is generated based on the input that it is given.

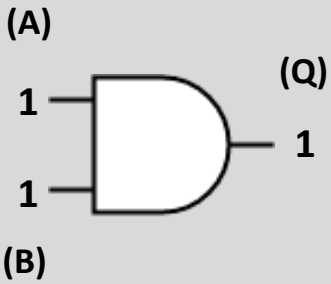


NOT Gate



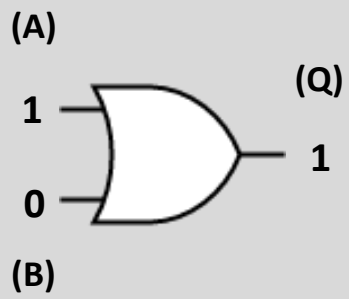
The NOT logic gate produces an output that is opposite to the input. Another way of saying this is that the number is **inverted**.
If the input is 1 (true) then the output is 0 (false). If the input is 0 (false) then the output is 1 (true).
This can also be written using a formula - **Q = NOT A** - A is the input, Q is the output.
It does not have to be A and Q, any letter can be used.

AND Gate



The AND logic gate has two inputs. In the example they are A and B. The two inputs produce a single output.
In plain English, the rule for the AND logic gate is "If both A and B are true, then the output is also true, otherwise it is false". This means that both inputs A and B have to be 1 (true) for the output Q to be 1 (true).
This can also be written using a formula - **Q = A AND B** - A and B are the inputs, Q is the output.

OR Gate



This OR logic gate also has two inputs. In the example above they are A and B. The two inputs produce a single output.

In plain English the rule for the OR logic gate is "If either or both A, B are true then the output is also true". This means that if A or B is 1 (true) then the output Q will be 1 (true).

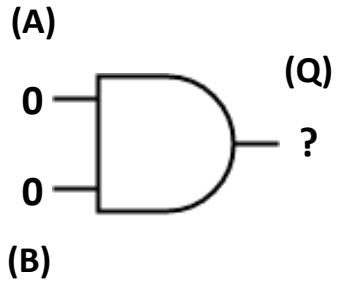
NOTE: With the OR gate, if both inputs are 1 (true) then the output will still be 1 (true). It would still meet the condition of 'either' input needing to be true.

This can also be written using a formula - **Q = A OR B** - A and B are the inputs, Q is the output.

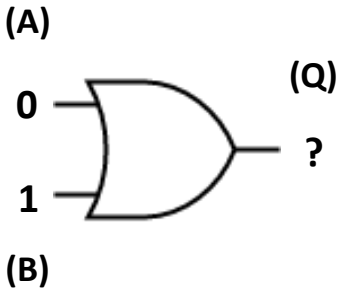
Exercise A

1 Find the output for the following logic gates based on the input given.

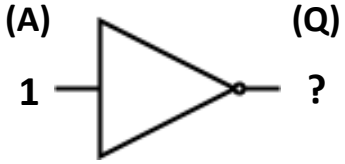
a)



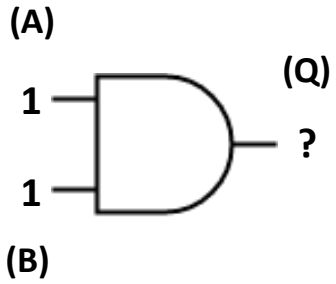
b)



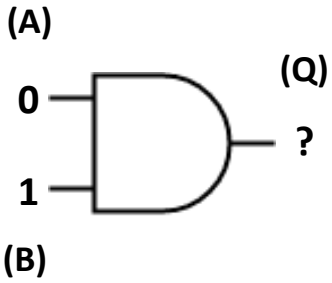
c)



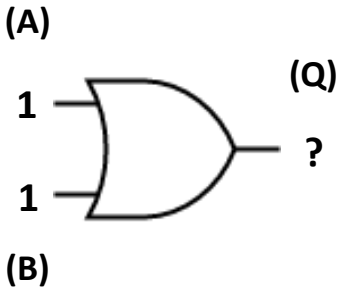
d)



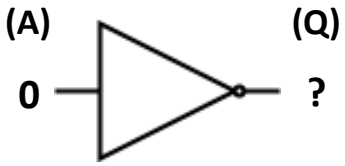
e)



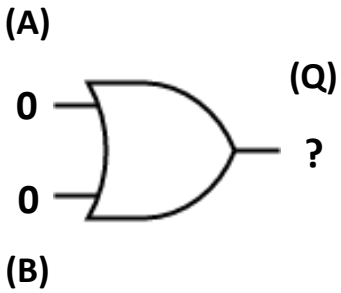
f)



g)



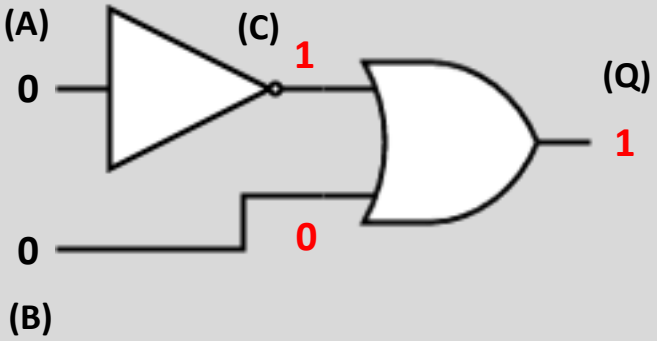
h)



When more than one logic gate is joined together it is known as a **logic circuit**.

The NOT, AND and OR gates still have the same effect on the input. The output will then move to the next gate and be used as the input.

Example



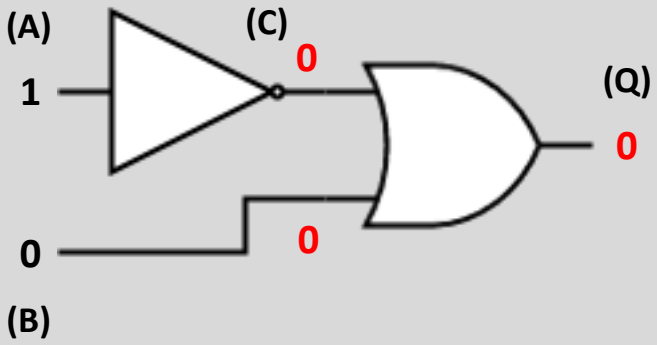
If both inputs (A) and (B) are 0 to begin with, the NOT gate would invert the 0 input to 1.

To make the logic circuits easier to follow, all outputs are usually labelled with a letter, so (C) would be 1.

The input (B) has not been changed so is still 0. The OR gate would have inputs (B) 1 and (C) 0, as one of the inputs is 1 the output would also be 1.

1 (true) would be output from this circuit.

Example



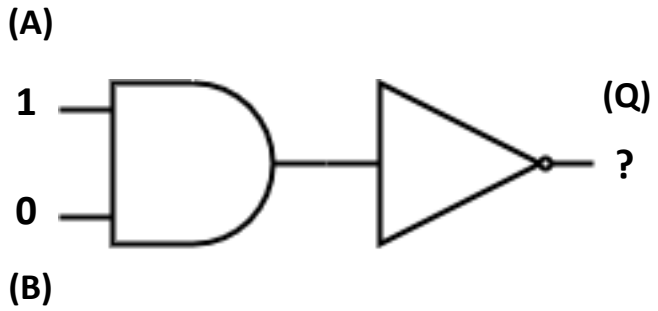
If input (A) is 1 and (B) is 0 to begin with, the NOT gate would invert the 1 input to 0.

At point (C) the value will be 0. Input (B) has not been changed so is still 0.

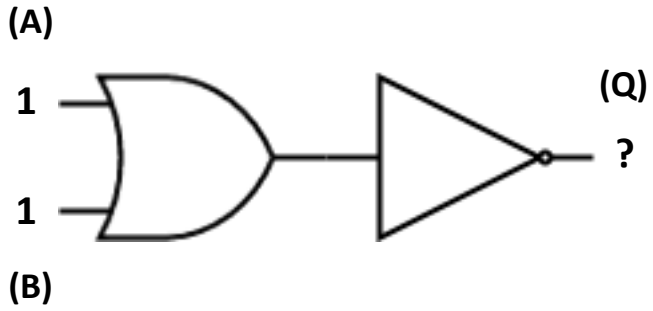
The OR gate would have inputs (B) 0 and (C) 0, as both of the inputs are 0 the output would also be 0.

2 Find the output for the following logic gates based on the input given.

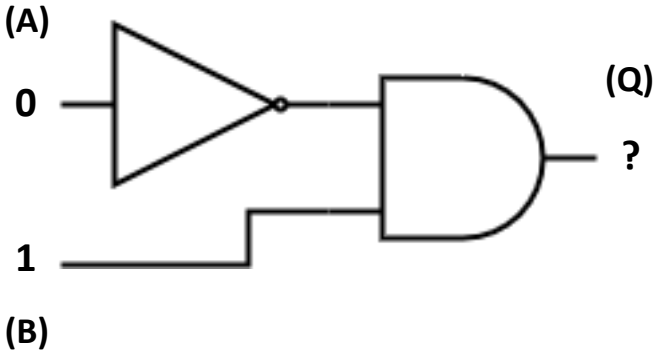
a)



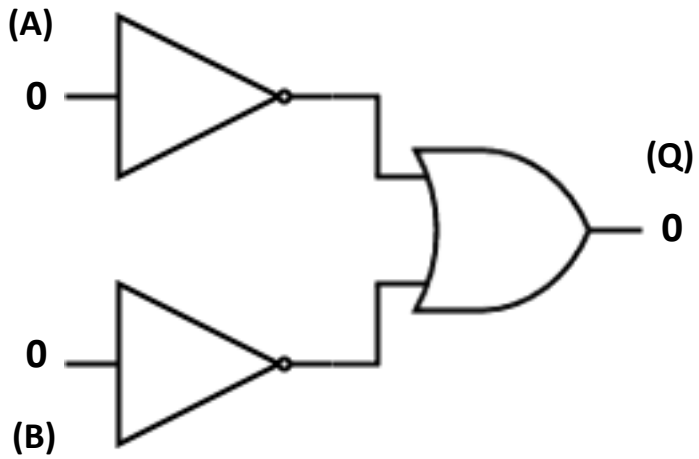
b)



c)



d)



Truth Tables

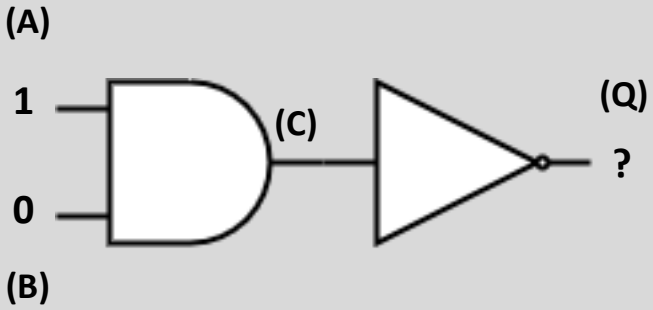
When working with logic circuits it is important that you are able to work out all of the different possible inputs and then all of the different possible outputs.

It would be very difficult to keep track of this in your head, especially with more than one input. For this we use a **truth table**.

A truth table lists every possible combination of inputs. From that you can then work out the different possible outputs.

Example

Input			Output (Q)
A	B	C	
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0



Firstly all of the different combinations of inputs are added. As there are two inputs there are 4 possible combinations – both 0, both 1, 1 and 0, and 0 and 1.

The inputs that you have listed are then used to work out the output to the logic circuit.

In the example above, (A) is 1 and (B) is 0. (C) will be 0 as both (A) and (B) need to be 1 for the AND gate to output 1.

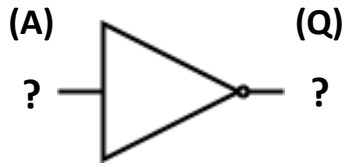
The input for the NOT gate will be (C) 0. The NOT gate will invert this to output 1 (true) from the circuit.

The same process will then be used to work out the remaining outputs.

Exercise B

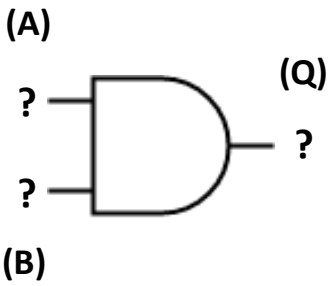
1 Draw a truth table for the following logic gates. Ensure that all of the different input combinations are included.

a)



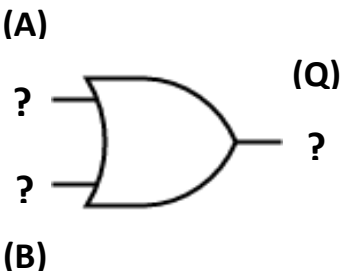
Input	Output
A	(Q)
1	
0	

b)



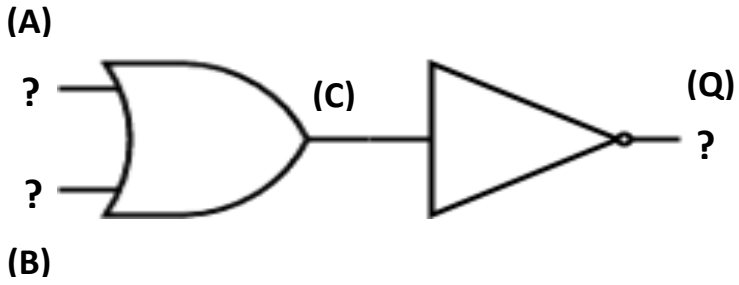
Input		Output
A	B	(Q)
1	1	
0	1	
1	0	
0	0	

c)



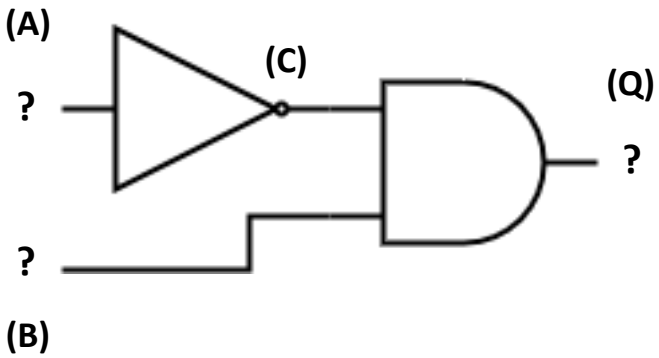
Input		Output
A	B	(Q)

2 Draw a truth table for the following logic circuit. Ensure that all of the different input combinations are included.



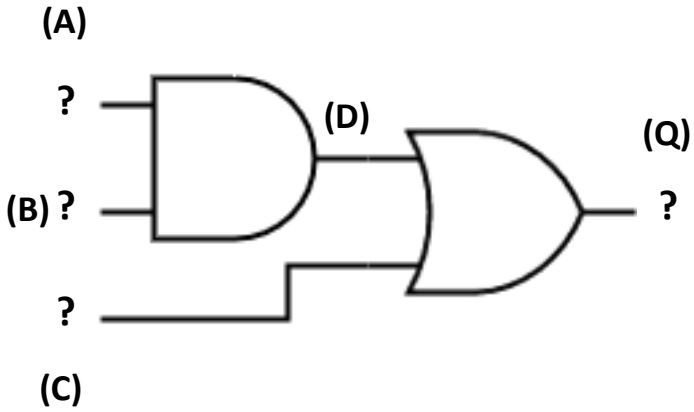
Input			Output
A	B	C	(Q)
1	1		
0	1		
1	0		
0	0		

3 Draw a truth table for the following logic circuit. Ensure that all of the different input combinations are included.



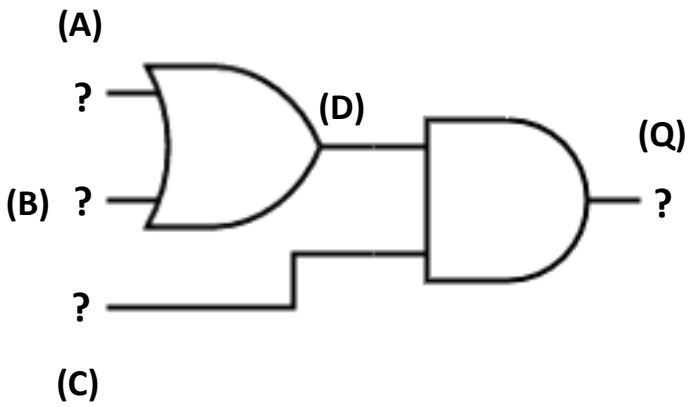
Input			Output (Q)
A	B	C	
1	1		
0	1		
1	0		
0	0		

4 Draw a truth table for the following logic circuit. Ensure that all of the different input combinations are included.



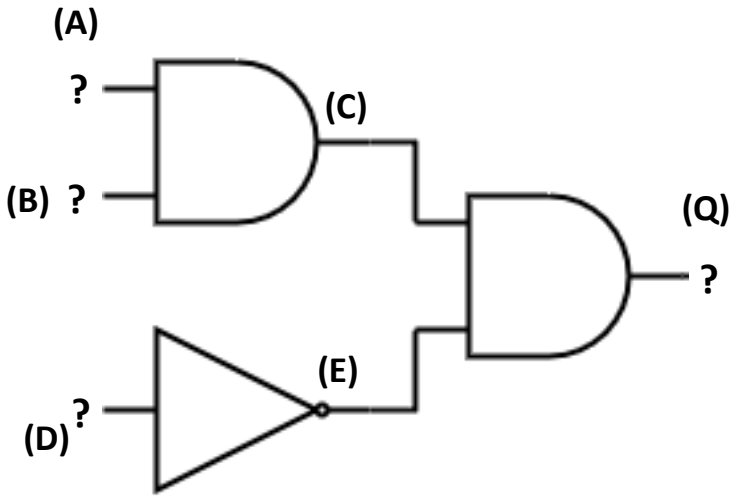
Input				Output (Q)
A	B	C	D	
1	1	1		
0	0	0		
1	0	0		
0	0	1		
1	1	0		
0	1	1		
1	0	1		
0	1	0		

5 Draw a truth table for the following logic circuit. Ensure that all of the different input combinations are included.



Input				Output
A	B	C	D	(Q)

6 Draw a truth table for the following logic circuit. Ensure that all of the different input combinations are included.



Input					Output
A	B	C	D	E	(Q)

Glossary

Terms

Term	Definition
Denary / Decimal	A number system which used the digits 0-9. The place value of each number is 10 times the number to the right.
Base 10	Another name for the denary / decimal number system.
Place Value	The value a number has based in its position within a number e.g. 1401 – 1 thousand, 4 hundreds and 1 unit.
Powers	Multiplying a number repeated by itself. Usually written above a number e.g. 3^2 would signify 3 x 3, 3^4 means 3 x 3 x 3 x 3
Binary	A number system which used the digits 0 & 1. The place value of each number is 2 times the number to the right.
Base 2	Another name for the binary number system.
Remainders	A number which is left over when two numbers do not divide equally.
Hexadecimal	A number system which used the symbols 1-9 & A-F. The place value of each number is 16 times the number to the right.
Base 16	Another name for the hexadecimal number system.
Nibble	A number which is 4 binary digits (bits) in length.
Left Shift	The name given to the process used to multiply a binary number. All the bits in the number are moved to the left.
Right Shift	The name given to the process used to divide a binary number. All the bits in the number are moved to the right.
Integer	A whole number
Modulo / Modulus	An operation that finds the remainder when one number is divided by another.
Quotient	The whole number result of dividing one number by another number. e.g. 20 / 6 = 3 r 2. The quotient is 3
Frequency	How frequently a number or letter appears in a sequence of numbers or letters e.g. 1124224 – 2 has a frequency of 3 as it appears 3 times.
Compression	The reduction in the number of bits needed to represent data. Reducing the size of a file.
Bit	A single binary digit, either 1 or 0.
Lossy	A form of compression where part of the data is removed and the data cannot be returned to its original state.
Lossless	A form of compression where no data is removed and the data can be returned back to its original state.

Decompressed or uncompressing	The act of turning a compressed file back into its original state.
Huffman encoding	A compression algorithm designed to reduce the size of a file through a reduction of bits.
Huffman tree	A diagram which shows the frequency of a value within a set of data, ready for Huffman encoding
Run-length encoding	A compression algorithm designed to reduce the size of a file by grouping repeated bits
Byte	The term given to 8 bits.
Run	The repetition of a letter or number when using run-length encoding.
Run count	The amount of times that a letter or number repeats when using run-length encoding.
Run value	The letter or number which is repeated when using run-length encoding.
Algorithm	A sequence of rules or instructions which can be followed to solve a problem or complete a task.
Variable	A piece of data which stored within an algorithm. The value of a variable will change within the algorithm.
Constant	A piece of data which stored within an algorithm. The value of a constant will not change within the algorithm.
String	A text string in an algorithm can only store letters, numbers, spaces and punctuation. They are usually shown in “” quotation marks.
Data	A value which is stored within a computer – in algorithms, values are stored within variables or constants.
Declaration	Where a variable is first written within an algorithm.
Assignment	Where a value is assigned to a variable or constant.
Substitution	Where the value stored inside a variable is changed for another value.
Condition	In an algorithm, a condition is something which needs to be met in order for something to happen e.g. IF age == 12.
Selection	Also known as **If Statements** or **conditional statements**, selection checks whether a condition is true e.g. IF age == 12 THEN…. An instruction is then selected based on this.
Nested selection	Where more than one selection statement is used inside each other. E.g. IF age = 12 THEN, IF height < 1.2m THEN..
Iteration	Repeating part or all of the instructions within an algorithm. Also known as a **Loop**.
Definite Iteration	An iteration which will repeat a fixed amount of times. Also known as a **For Loop**.
Indefinite Iteration	An iteration which will repeat while a condition is true. Also known as **While Loop**,

Subroutine	Subroutines are used in algorithms where part of the algorithm needs to be repeated multiple times.
Efficient	In algorithms, efficiency is using the fewest instructions possible to complete a task.
Executed	Another word for instructions being worked through or run.
Parameters	The data which is passed to a subroutine when it is called.
Search algorithm	A set of instructions that can be followed to find a piece of data within a list e.g. find 9 from 1, 3, 6, 8, 9, 14, 17.
Binary Search	An efficient search algorithm that finds an item in an ordered list
Sort algorithm	A set of instructions that can be followed to sort a list of data in a particular order e.g. smallest to largest e.g. sort 6, 7, 13, 2, 8, 10 to 2, 6, 7, 8, 10, 13.
Bubble sort	A sort algorithm which sorts data into order – normally size order
Pass	When performing a Bubble sort, each time that you have gone through the list is known as a pass.
Trace table	A table which allows you to track the values of variables as they change throughout an algorithm.
Dry run	Running through an algorithm using a method such as a trace table to ensure that the algorithm works or to spot errors.
Logic gate	Are used to make decisions based on the data which they receive. This data is in the form of electronic signals, which can be 1 'on' or 0 'off'. When using logic gates 1 is also referred to as 'true' and 0 as 'false'.
Logic circuits	When more than one logic gate is joined together. Logic circuits are used to make decisions on the inputs that they are given.
Boolean	A Boolean value only has two possible values: true or false.
CPU	The Central Processing Unit. Processes all instructions inside a computer.
Inverted	Turned to be the opposite e.g. turn 1 to 0 or 0 to 1.
NOT gate	A gate which inverts its input e.g. turns 1 (true) to 0 (false) or 0 (false) to 1 (true).
AND gate	A gate where both inputs must be 1 (true) for the output to be 1 (true).
OR gate	A gate where either input can be 1 (true) for the output to be 1 (true). Both inputs can also be 1 (true).
Truth Table	Allows you to work out every combination of inputs and the outputs which they generate from a logic circuit.

Aithmetic Operators

Symbol	Name	Example
+	Addition	`print 6 + 2` `>>> 8`
-	Subtraction	`x = 3 - 2` `>>> 1`
*	Multiplication	`x = 5 * 2` `>>> 1`
/	Division	`x = 16 / 8` `>>> 2`
% or MOD	Modulus	`x = 5 % 3` `>>> 2`
DIV	Quotient	`x = 17 DIV 5` `>>> 3`

Relational Operators

Symbol	Name	Example

==	Equal to	`4 == 4`
		`>>> True`
		
		`8 == 4`
		`>>> False`

≠ or !=	Not equal to	`7 != 4`
		`>>> True`
		
		`7 == 7`
		`>>> False`

<	Less than	`4 < 8`
		`>>> True`
		
		`6 < 4`
		`>>> False`

≤ or <=	Less than or equal to	`4 <= 4`
		`>>> True`
		
		`2 <= 4`
		`>>> True`
		
		`8 <= 4`
		`>>> False`

>	Greater than	`8 > 4`
		`>>> True`
		
		`4 > 4`
		`>>> False`

≥ or >=	Greater than or equal to	`4 >= 4`
		`>>> True`
		
		`4 >= 2`
		`>>> True`
		
		`4 >= 8`
		`>>> False`