OCR
RECOGNISING ACHIEVEMENT

Exemplar Work for SAMs

Units A452 and A453

GCSE Computing Controlled Assessment

Unit A452 Practical Investigation

Unit Recording Sheet

OCR RECOGNISING ACHIEVEMENT

Please read the instructions printed on the other side of this form. **One** of these Unit Recording Sheets, suitably completed, should be attached to the assessed work of **each** candidate.

Unit	**A452**		**Year**	
Centre Name			**Centre Number**	
Candidate Name			**Candidate Number**	

	Guidance			**Teacher Comment**	**Mark**
Practical Investigation	There may be little or no evidence of any practical investigation. The evidence supplied is minimal and poorly documented with little relevance to the set task. The practical evidence may all be the result of group or teacher led activity with little input from the student. **[0 - 5]**	There is evidence of a practical investigation The evidence supplied is documented clearly and is relevant to the set task There is evidence of individual research beyond the group activity and any teacher led activity. The practical investigations show signs of planning but there may be omissions made in assessing the consequences. **[6 - 10]**	There is evidence of a well structured practical investigation The evidence supplied is well organised and clearly relevant to the set task There is extensive evidence of individual practical investigation beyond the group activity and any teacher led activity The practical investigation shows clear signs of planning and a structured approach to evidence gathering to provide a complete investigation of the set topic area and beyond. Practical investigation has been carried out with skill and due regard to safety issues. **[11 - 15]**	Decent attempt to investigate LMC. Some examples used and explained well. Attempts ate the subsequent taskss OK. Overall lacking in detail and explanation of the process.	8 **Max 15**

Effective and efficient use of techniques	The techniques used may not be entirely appropriate to the problem and will only produce partially working solutions to a small part of the problem. **[0 - 3]**	The techniques will be used appropriately giving working solutions to most of the parts of the problem. Some parts of the solution may be executed in a partial or inefficient manner. **[4 - 7]**	The techniques are used appropriately in all cases giving an efficient, working solution for all parts of the problem. **[8 - 10]**	The techniques are largely used adequately but not always effeiciently. most of the solutions work but there is a lack of explanation throughout the development. The solutions are not always efficient.	6 **Max 10**
Technical understanding	The candidate demonstrates a limited understanding of the technical issues related to the scenario. Little detail is presented. There will be limited indication of any evidence provided being analysed. There is little correct use of technical terminology. **[0 - 3]**	The candidate demonstrates a reasonable understanding of the technical issues related to the scenario. The amount of detail presented is adequate to support the arguments. There is some analysis carried out on the evidence collected. The use of technical terminology is largely correct but it may be limited. **[4 - 7]**	The candidate demonstrates a thorough and secure understanding of the technical issues related to the scenario. A wide range of relevant and detailed information is presented. The evidence which has been collected is fully analysed. Technical terminology is used correctly. At the top end of the band, this will be extensive and confidently used. **[8 - 10]**	The use of technical terms is reasonable but not always accurate and the report lacks deatil and analysis.	5 **Max 10**

Conclusions and evaluation	Conclusions are weak or missing with little or no justification. The solution is presented with little if any evidence of testing. The evidence of written communication is limited with little or no use of specialist terms. There are many errors in spelling, punctuation and grammar. Information may be ambiguous or disorganised. There is limited if any reference to evidence. The evaluation may be simplistic with little or no relevance. **[0 - 3]**	The material has structure and coherence with justifiable conclusions being reached although there may be some omissions. There is evidence that the solutions have been tested for basic functionality. Candidates will have produced a sound evaluation which reviews some aspects of the task. Evidence of good written communication using some specialist terms. There are few errors in spelling, grammar and punctuation. Information for the most part will be presented in a structured format. Specialist terms will be used appropriately and for the most part correctly. **[4 – 7]**	Thorough and convincing conclusions have been reached, which are borne out by the research carried out by the candidate. The solutions are fully tested and there is little doubt that the solutions presented are fully functional. This material has been presented in a clear and relevant way which is simple to navigate. A high level of written communication is obvious throughout the task and specialist terms/technology with accurate use of spelling is used. Grammar and punctuation is consistently correct. Information is presented in a coherent and structured format. The evaluation will be relevant, clear, organised and presented in a structured and coherent format. **[8- 10]**	There is evidence of some testing for function throughout the report but this lacks organisation and is not complete. The evaluation is minimal and adds little.	4 **Max 10**
				Total/45	**23**

GCSE Computing

Unit A452: Practical investigation

Exemplar Material for A452 SAM

INSIDE THE MACHINE

Most computers are built to the same basic architecture – the Von Neumann architecture. They have **memory** where program instructions and other data are stored and they have a **processor** that decodes and carries out the program instructions.

The processor has special memory locations called registers. This is where the program instructions are acted on. There is a working demonstration of how the processor and memory interact called the Little Man Computer (LMC). Some versions run as an embedded applet in a browser. The details are here:http://www.atkinson.yorku.ca/~sychen/research/LMC/LMCHome.html

The applet itself is here:
http://www.atkinson.yorku.ca/~sychen/research/LMC/LittleMan.html

Alternatively you can access another version from:
http://www.cs.ru.nl/~erikpoll/Teaching/III/lmc/

1 Investigate the instruction set provided with one implementation of the LMC.

2 Load and run at least two of the demonstration programs supplied with the implementation.

3 Explain in your own words what happens as each of the instructions is executed.

4 Write programs to run in LMC:

- **(i)** Take in two numbers and output the smaller first, then the larger
- **(ii)** Produce a multiplication table from 1 to 10 for any number input by the user
- **(iii)** Input five numbers and output them in reverse order.

Produce evidence to show that you have planned, written and tested your code.

5 Produce an evaluation of your solutions.

6 Write a conclusion about the possibility of writing effective and complex programs with only a limited instruction set.

The Little Man Computer

Task 1 Investigate the instruction set provided with one implementation of the LMC.

The Little Man Computer is like a real computer but not as powerful. It can only do some of the things that a real computer can do. The instruction set is a list of the commands you can use with it. It has ten of these and they let you move data, add, subtract and check what is happening after you do something.

LDA: load the accumulator with something.
STA: store what is in the accumulator in memory.
ADD: add data from memory into the accumulator.
SUB: subtract data in memory from the accumulator.
INP: input a number – it goes into the accumulator.
OUT: output what is in the accumulator.
BRZ: branch to the place indicated if there is a zero in the accumulator.
BRP: branch to the place indicated if the number if the accumulator has a zero or a positive number in it.
BRA: branch anyway
HLT: halt the program
DAT: this shows you where data is kept.

These instructions can be given either as machine code numbers or as assembly language mnemonics.

I tried some of the instructions in the LMC.

```
INP
STA NUM
INP
SUB NUM
OUT
HLT
NUM DAT
```

This program takes two inputs and subtracts the second one from the first. Here is the LMC when the program has been compiled.

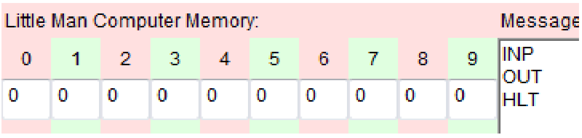
You can see that INP has been changed to 901 and stored in cell 0.
STA has become 3 and the 06 means that the data is to be stored in location 6. Location 6 is the place where the compiler has decided that NUM must go.
Cell 2 has another INP or 901 in it. Cell 3 has 2 for subtract and 06 for where it must look to get the number to subtract.
902 in cell 4 outputs the result from the accumulator.
0 in 5 is the halt instruction.

Task 2. Load and run at least two of the demonstration programs supplied with the implementation.

I first tried this example from the web site.

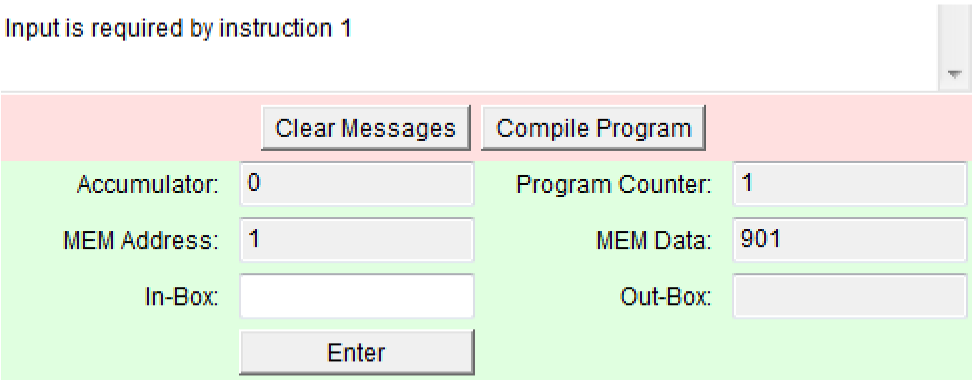
Program 1
INP
OUT
HLT

Here it is in the LMC.

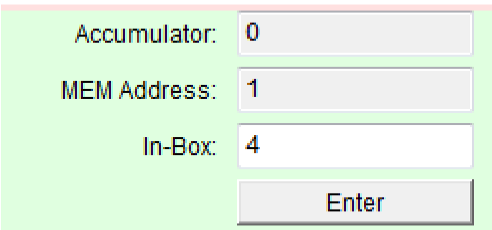


All this program does is to take a number from the user and output it. Here is evidence that I did that.

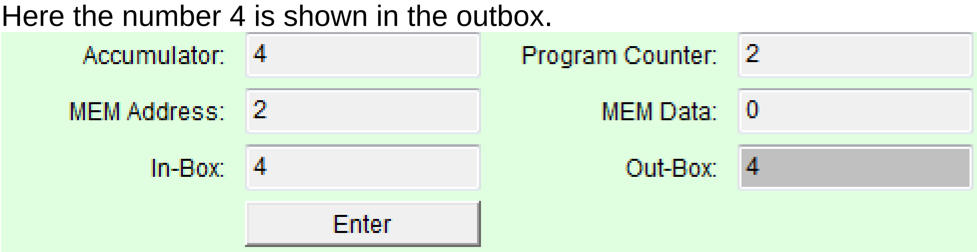
Running the program
When the run button is clicked, the program counter is set to 1 and you can see that input is required.



I entered the number 4.



Here the number 4 is shown in the outbox.



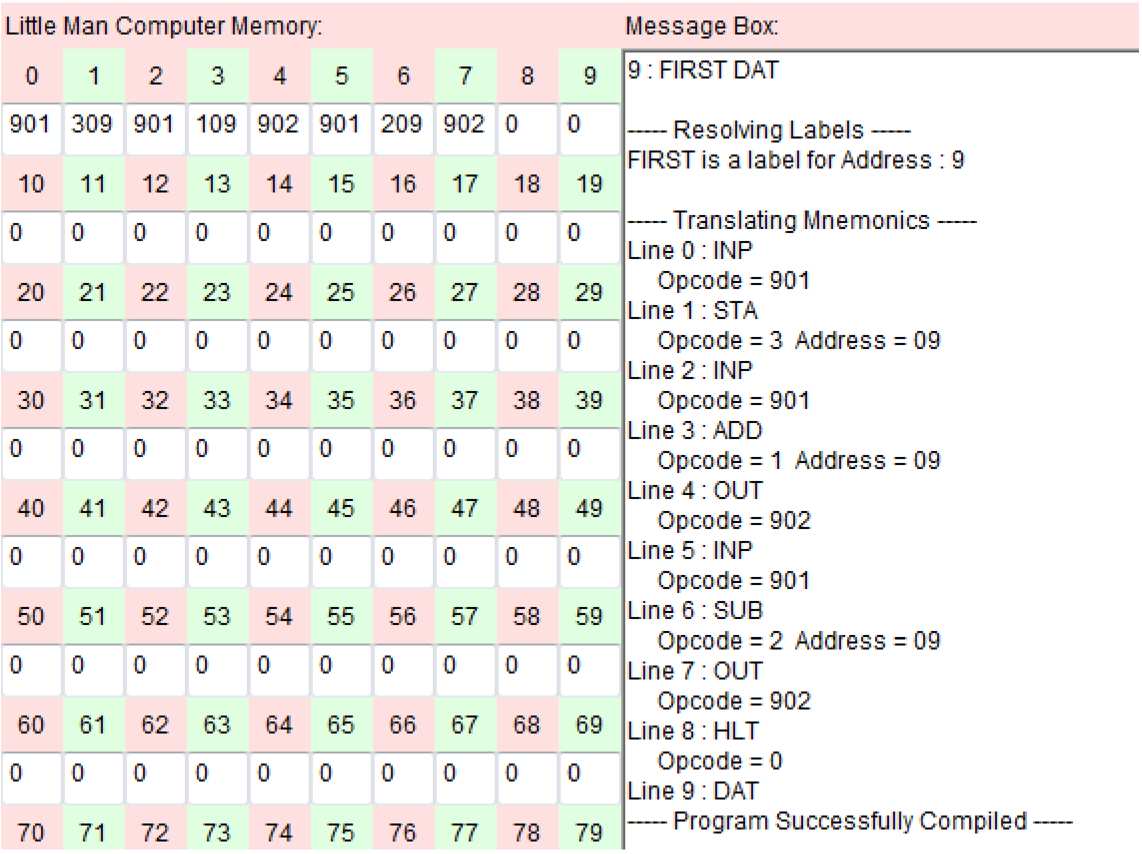
The third instruction is HLT so the program ends.

I then tried another program from the web site. It is supposed to add then subtract numbers.

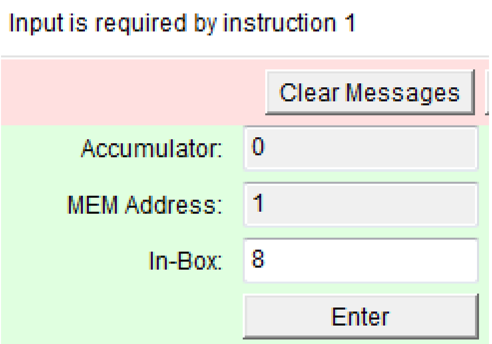
Program 2

INP
STA FIRST
INP
ADD FIRST
OUT
INP
SUB FIRST
OUT
HLT
FIRST DAT

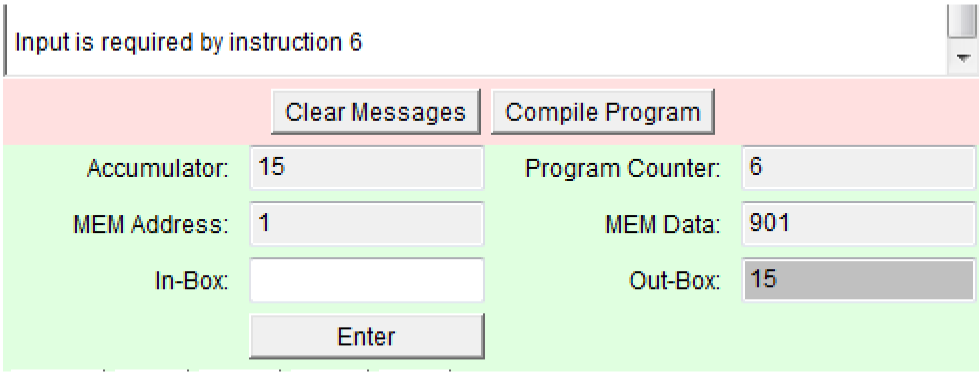
Here it is compiled.



I then ran it. Here is my first input which is 8.

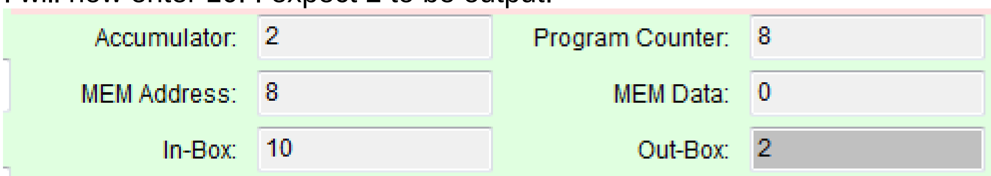


I then entered 7. I expected the answer 15 to be output.



Here you can see 15 in the out-box.

I will now enter 10. I expect 2 to be output.



Task 3. Explain in your own words what happens as each of the instructions is executed.

Program 1

INP
This takes a number from the user and puts it in the accumulator.

OUT
This outputs the value in the accumulator.

HLT
This stops the program.

Program 2

INP
This takes a number from the user and puts it in the accumulator. In my test, this is 8.

STA FIRST
This stores that number in the memory cell labelled FIRST.

INP
This takes a number from the user and puts it in the accumulator. In this case, I used 7.

ADD FIRST
This adds the number in FIRST to whatever is in the accumulator, which is the number last entered. 8+7=15 so that goes into the accumulator.

OUT
This outputs what is in the accumulator, which is the result of the addition. In this case, it is 15.

INP
This takes a number from the user and puts it in the accumulator. I used 10 in this case.

SUB FIRST
This takes away the number in FIRST from whatever is now in the accumulator. 10-8=2 so 2 is now in the accumulator.

OUT
This outputs the result, which in my test was 2.

HLT
This halts the program.

FIRST DAT
This assigns a label FIRST to a memory location which is used to store the first value input in this program.

Task 4. Write programs to run in LMC:

i. Take in two numbers and output the smaller first, then the larger.

Produce evidence to show that you have planned, written and tested your code.

Here is the pseudocode to solve this problem.

Get the first number
Store it as FIRST
Get the second number
Store it as SECOND
Subtract the first number (the second is still in the accumulator)
If the result is positive output the second number then the first
Otherwise output the first number followed by the second.

If we store both numbers, then we can bring them back to output them.

Here is the LMC code that achieves this.

```
INP
STA FIRST
INP
STA SECOND
SUB FIRST
BRP SECONDBIG
LDA SECOND
OUT
LDA FIRST
OUT
BRA PROGEND
SECONDBIG LDA FIRST
OUT
LDA SECOND
OUT
PROGEND HLT
FIRST DAT
SECOND DAT
```

Here I run it with 50 followed by 60. I expect the numbers to come out in the same order.

Accumulator:	0	Program Counter:	1
MEM Address:	1	MEM Data:	901
In-Box:	50	Out-Box:	

Here is the output box after I ran this.

Accumulator:	60	Program Counter:	15
MEM Address:	15	MEM Data:	0
In-Box:	60	Out-Box:	60

It worked correctly, although you cannot see the 50 because it happened very quickly.

I will now test it the other way round. I will enter 100 then 20. I expect the 20 to come out first then the 100.

This worked too. Here is the 100 showing at the end.

Accumulator:	100	Program Counter:	15
MEM Address:	15	MEM Data:	0
In-Box:	20	Out-Box:	100

The output happens quickly so it is best to run it using the slow button.
Here is the output when tested with 3 then 4. The number 4 is the last output.

Clear Messages | Compile Program

Accumulator:	4	Program Counter:	15
MEM Address:	15	MEM Data:	0
In-Box:	4	Out-Box:	4

Here we test it with 4 then 3. Again 4 is the last output with 3 being visible as the last input into the in-box.

Accumulator:	4	Program Counter:	15
MEM Address:	15	MEM Data:	0
In-Box:	3	Out-Box:	4

ii. Produce a multiplication table from 1 to 10 for any number input by the user

Produce evidence to show that you have planned, written and tested your code.

This is hard to do with LMC because it can't do multiplication. I have found out that to get round that, you have to add the numbers together as many times as you need

So, to make this happen, what we need to do first is to get the number from the user, then add it to itself as many times as needed. I will first test this by writing a program to multiply a number by 3. I will put the 3 in a memory location and then add the input number together 3 times.

I will set up a counter to count how many times this has happened.

Pseudocode

Get number
Store number
Add number
Get counter
Add 1
Get THREE
Take away counter
If positive go back and do another addition
Output result

```
INP
STA NUMBER
LOOP LDA TOTAL
ADD NUMBER
STA TOTAL
LDA COUNTER
ADD ONE
STA COUNTER
LDA THREE
SUB COUNTER
BRP LOOP
LDA TOTAL
OUT
HLT

THREE DAT 003
COUNTER DAT 001
ONE DAT 001
NUMBER DAT
TOTAL DAT
```

I tested this by entering the number 4. I expected this to be multiplied by 3 and give the answer 12.

Accumulator:	12	Program Counter:	13
MEM Address:	13	MEM Data:	0
In-Box:	4	Out-Box:	12

You can see that it worked. Just to be sure, I will test it with 5. I expect the answer 15.

Accumulator:	15	Program Counter:	13
MEM Address:	13	MEM Data:	0
In-Box:	5	Out-Box:	15

That worked too.

iii. Input five numbers and output them in reverse order.

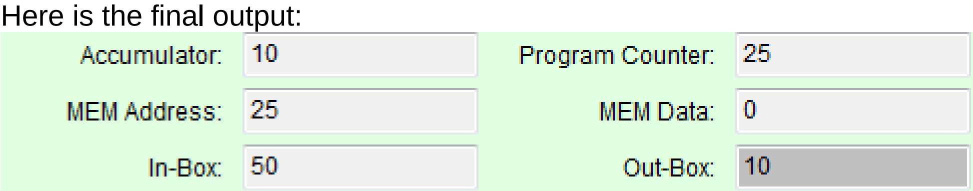
Produce evidence to show that you have planned, written and tested your code.

This is quite easy to do. You take the five numbers, store them in separate locations, then output them in whatever order you want. In this case it will be in reverse order.

Here is the LMC code.

```
INP
STA ONE
INP
STA TWO
INP
STA THREE
INP
STA FOUR
INP
STA FIVE

LDA FIVE
OUT
LDA FOUR
OUT
LDA THREE
OUT
LDA TWO
OUT
LDA ONE
OUT

ONE DAT
TWO DAT
THREE DAT
FOUR DAT
FIVE DAT
```

I tested this by inputting 10, 20, 30, 40, 50. I expected the output to be 50, 40, 30, 20, 10.
Here is the final output:



You can see that the in-box still has the 50 in it but the out-box has finished on 10, which is correct.

5. Produce an evaluation of your solutions.

I am pleased with what I did and most of the solutions work well. I didn't manage to finish the multiplication because I ran out of time.

6. Write a conclusion about the possibility of writing effective and complex programs with only a limited instruction set.

It would be hard to write something big in LMC. Even comparing two numbers took ages to do. It would be better if it had more instructions and a proper if.. then. So, I think that if you wanted to do something complex like write a word processor, it would take too long.

GCSE Computing Controlled Assessment

Unit A452 Practical Investigation

Unit Recording Sheet

Please read the instructions printed on the other side of this form. **One** of these Unit Recording Sheets, suitably completed, should be attached to the assessed work of **each** candidate.

Unit	**A452**		**Year**	
Centre Name	sample		**Centre Number**	
Candidate Name	sample A		**Candidate Number**	

	Guidance			**Teacher Comment**	**Mark**
Practical Investigation	There may be little or no evidence of any practical investigation. The evidence supplied is minimal and poorly documented with little relevance to the set task. The practical evidence may all be the result of group or teacher led activity with little input from the student. **[0 - 5]**	There is evidence of a practical investigation The evidence supplied is documented clearly and is relevant to the set task There is evidence of individual research beyond the group activity and any teacher led activity. The practical investigations show signs of planning but there may be omissions made in assessing the consequences. **[6 - 10]**	There is evidence of a well structured practical investigation The evidence supplied is well organised and clearly relevant to the set task There is extensive evidence of individual practical investigation beyond the group activity and any teacher led activity The practical investigation shows clear signs of planning and a structured approach to evidence gathering to provide a complete investigation of the set topic area and beyond. Practical investigation has been carried out with skill and due regard to safety issues. **[11 - 15]**	Some good evidence of investigation beyond the initial starting point but, while qquite good, it does lack some depth of treatment. The tasks are carried out with some planning including evidence of flow charts and some detail of the approach taken. Not all the choices made are clearly explained.	12 **Max 15**

Effective and efficient use of techniques	The techniques used may not be entirely appropriate to the problem and will only produce partially working solutions to a small part of the problem. **[0 - 3]**	The techniques will be used appropriately giving working solutions to most of the parts of the problem. Some parts of the solution may be executed in a partial or inefficient manner. **[4 - 7]**	The techniques are used appropriately in all cases giving an efficient, working solution for all parts of the problem. **[8 - 10]**	Effective solutions and quite efficient but lacks explanation in some places.	8 **Max 10**
Technical understanding	The candidate demonstrates a limited understanding of the technical issues related to the scenario. Little detail is presented. There will be limited indication of any evidence provided being analysed. There is little correct use of technical terminology. **[0 - 3]**	The candidate demonstrates a reasonable understanding of the technical issues related to the scenario. The amount of detail presented is adequate to support the arguments. There is some analysis carried out on the evidence collected. The use of technical terminology is largely correct but it may be limited. **[4 - 7]**	The candidate demonstrates a thorough and secure understanding of the technical issues related to the scenario. A wide range of relevant and detailed information is presented. The evidence which has been collected is fully analysed. Technical terminology is used correctly. At the top end of the band, this will be extensive and confidently used. **[8 - 10]**	There is evidence of good understanding of the technical aspects and basic features are used effectively to demonstrate a good understanding, but explanations lack the deatil that would demonstrate a full understanding.	9 **Max 10**

Conclusions and evaluation	Conclusions are weak or missing with little or no justification. The solution is presented with little if any evidence of testing. The evidence of written communication is limited with little or no use of specialist terms. There are many errors in spelling, punctuation and grammar. Information may be ambiguous or disorganised. There is limited if any reference to evidence. The evaluation may be simplistic with little or no relevance. **[0 - 3]**	The material has structure and coherence with justifiable conclusions being reached although there may be some omissions. There is evidence that the solutions have been tested for basic functionality. Candidates will have produced a sound evaluation which reviews some aspects of the task. Evidence of good written communication using some specialist terms. There are few errors in spelling, grammar and punctuation. Information for the most part will be presented in a structured format. Specialist terms will be used appropriately and for the most part correctly. **[4 – 7]**	Thorough and convincing conclusions have been reached, which are borne out by the research carried out by the candidate. The solutions are fully tested and there is little doubt that the solutions presented are fully functional. This material has been presented in a clear and relevant way which is simple to navigate. A high level of written communication is obvious throughout the task and specialist terms/technology with accurate use of spelling is used. Grammar and punctuation is consistently correct. Information is presented in a coherent and structured format. The evaluation will be relevant, clear, organised and presented in a structured and coherent format. **[8- 10]**	A decent attempt to provide evidence but the testing is limited in some cases, otherwise well organised with a good evaluation of the topic.	8 **Max 10**
				Total/45	**37**

GCSE Computing

Unit A452: Practical investigation

Exemplar Material for A452 SAM

INSIDE THE MACHINE

Most computers are built to the same basic architecture – the Von Neumann architecture. They have **memory** where program instructions and other data are stored and they have a **processor** that decodes and carries out the program instructions.

The processor has special memory locations called registers. This is where the program instructions are acted on. There is a working demonstration of how the processor and memory interact called the Little Man Computer (LMC). Some versions run as an embedded applet in a browser. The details are here:http://www.atkinson.yorku.ca/~sychen/research/LMC/LMCHome.html

The applet itself is here:
http://www.atkinson.yorku.ca/~sychen/research/LMC/LittleMan.html

Alternatively you can access another version from:
http://www.cs.ru.nl/~erikpoll/Teaching/III/lmc/

1 Investigate the instruction set provided with one implementation of the LMC.

2 Load and run at least two of the demonstration programs supplied with the implementation.

3 Explain in your own words what happens as each of the instructions is executed.

4 Write programs to run in LMC:

(i) Take in two numbers and output the smaller first, then the larger

(ii) Produce a multiplication table from 1 to 10 for any number input by the user

(iii) Input five numbers and output them in reverse order.

Produce evidence to show that you have planned, written and tested your code.

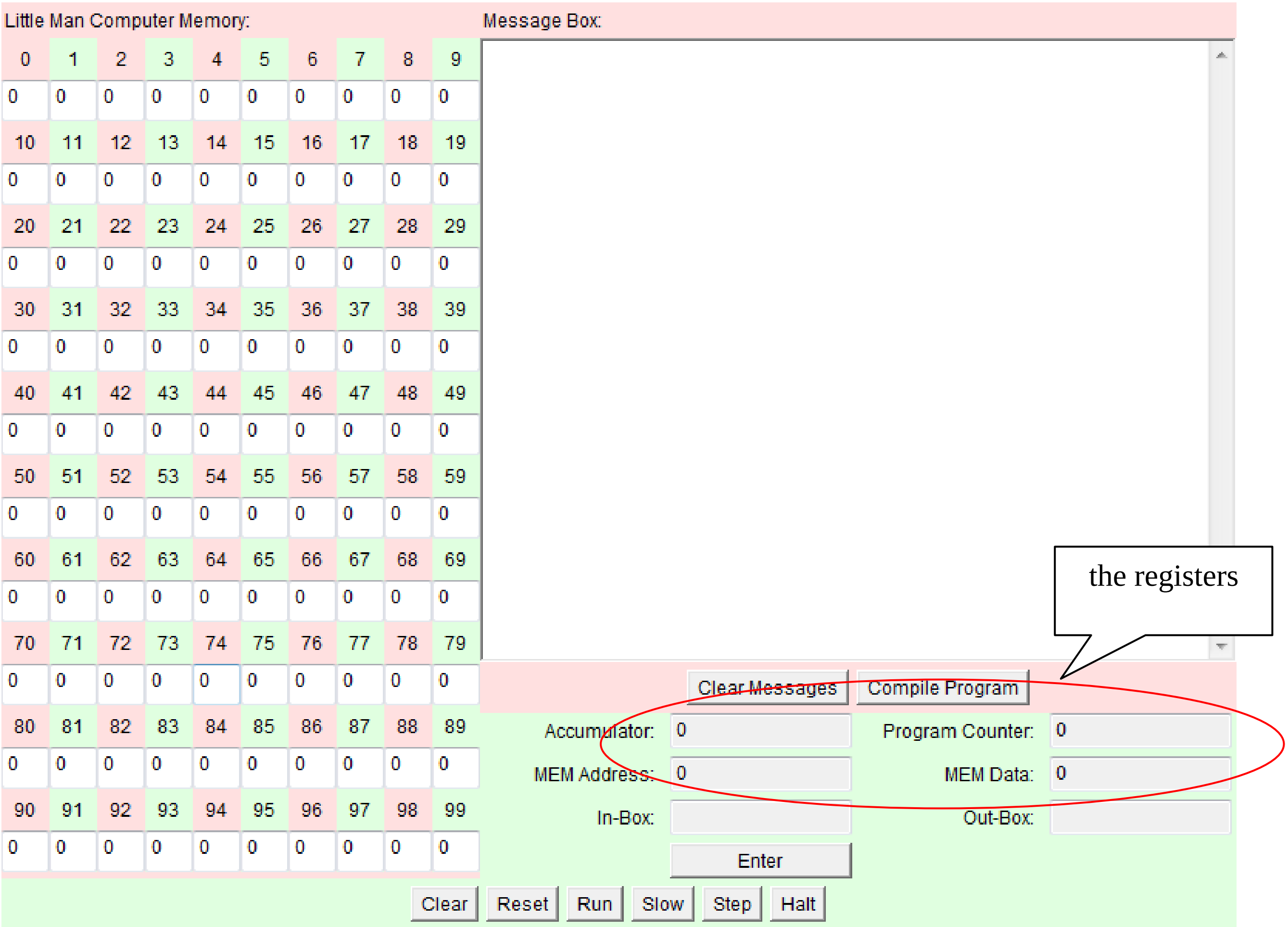
5 Produce an evaluation of your solutions.

6 Write a conclusion about the possibility of writing effective and complex programs with only a limited instruction set.

The Little Man Computer

Task 1. Investigate the instruction set provided with one implementation of the LMC.

The Little Man Computer is a simulation of what happens in a real computer. It is a much reduced version because it only has ten instructions in its instruction set instead of over a hundred in a typical PC. Also, it only has four registers against over twenty in modern PCs. The registers are the accumulator, the program counter, the memory address register and the memory data register.



This shot shows the start up screen before a program has been entered. A program can be typed or pasted into the message box in assembly language mnemonics and then compiled to produce the machine code by clicking the button.

The instruction set

Here are the ten instructions together with their assembly language and machine code equivalents. Although there are only ten, they can be combined in a program to do many useful tasks.

Instruction	Mnemonic	Code	What it means
LOAD	LDA	5	Copy a number into the accumulator.
STORE	STA	3	Store a number at a stated address
ADD	ADD	1	Add a number from a stated address to whatever is in the accumulator.
SUBTRACT	SUB	2	Subtract the number at a stated address from the accumulator.
INPUT	INP	901	Take an input from the in-box and put it in the accumulator.
OUTPUT	OUT	902	Output what is in the accumulator to the out-box.
BRANCH IF ZERO	BRZ	7	Branch to the stated address if zero is in the accumulator.
BRANCH IF ZERO OR POSITIVE	BRP	8	Branch to the stated address if zero or a positive number is in the accumulator.
BRANCH ALWAYS	BRA	6	Branch to the stated address without checking the accumulator.
END	HLT	000	End the program.
DATA LOCATION	DAT	(the data)	This marks where a particular item of data is to be found. It allows you to label it.

These instructions can be given either as machine code numbers or as assembly language mnemonics.

Example instructions in use

Here are some examples of how the instruction mnemonics can be used.

LDA 12: load the accumulator with whatever is in location 12.
STA 20: store whatever is in the accumulator in location 20.
ADD 20: add whatever is in location 20 to whatever is in the accumulator.
SUB 20: subtract whatever is in location 20 from the accumulator.
INP: this just waits for an input then copies it into the accumulator, replacing whatever is already in there.
OUT: this copies whatever is in the accumulator to the out-box.
BRZ 20: branch to the instruction in location 20 if the accumulator contains zero.
BRP 20: branch to the instruction in location 20 if the accumulator contains zero or any positive number.
BRA 20: unconditionally branch to the instruction in location 20.
HLT: stop the program.

Task 2. Load and run at least two of the demonstration programs supplied with the implementation.

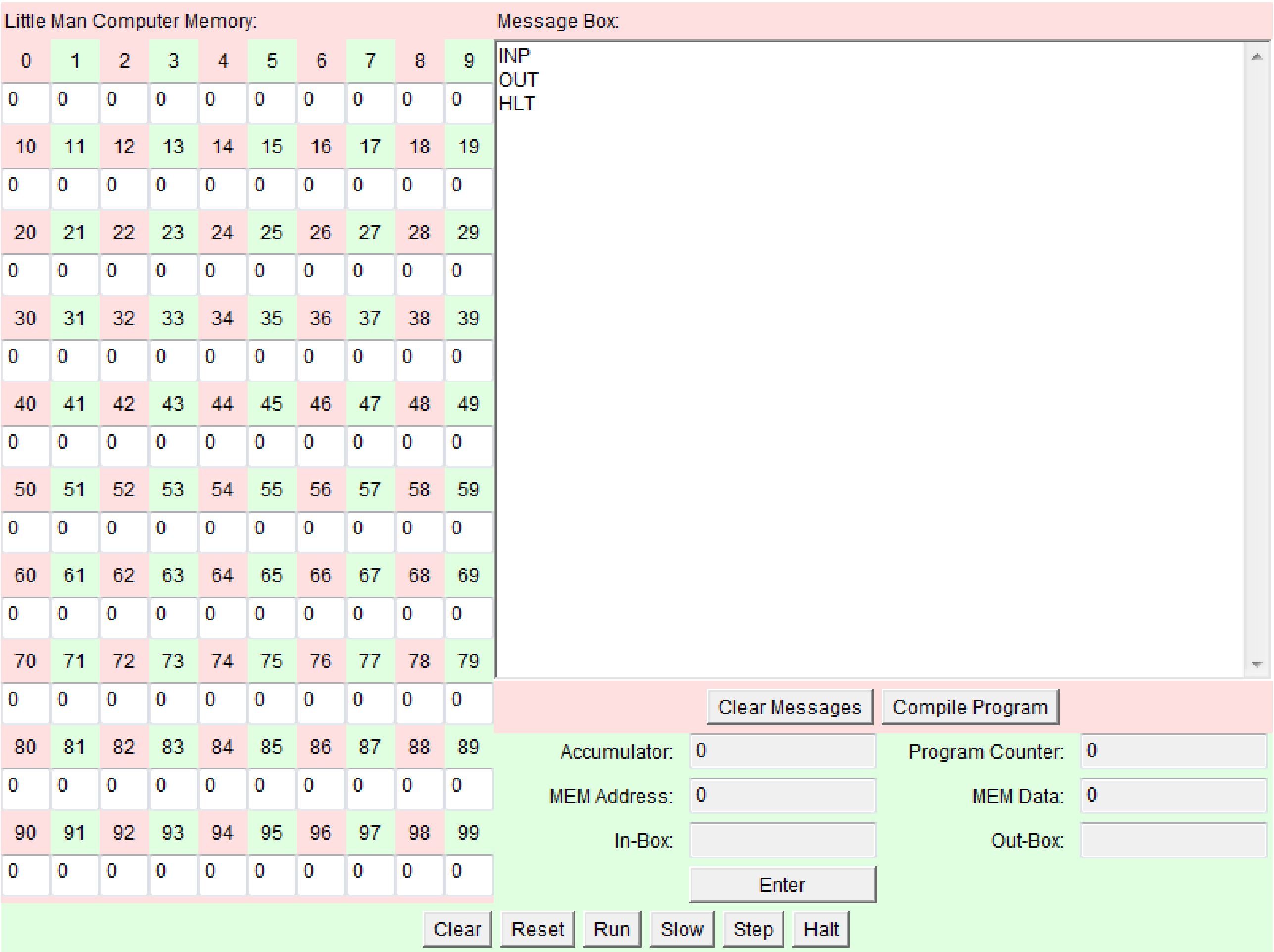
AND

Task 3. Explain in your own words what happens as each of the instructions is executed.

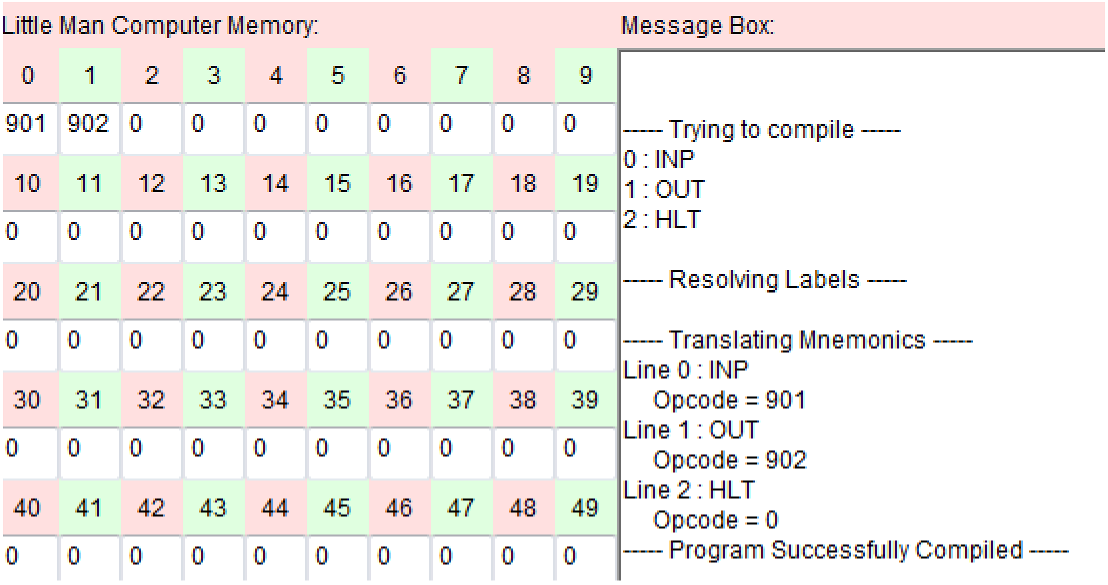
Here is the simplest one that is on the website.

INP
OUT
HLT

Here it is in the message box.



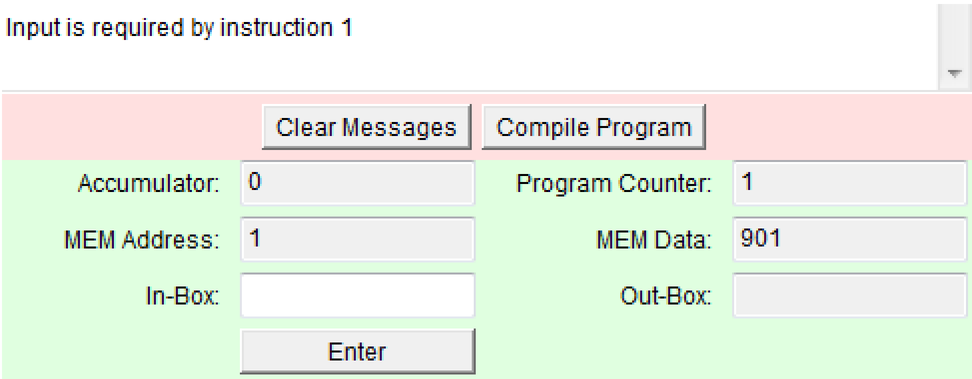
When the compile button is clicked, the program is converted into machine code and placed in memory.



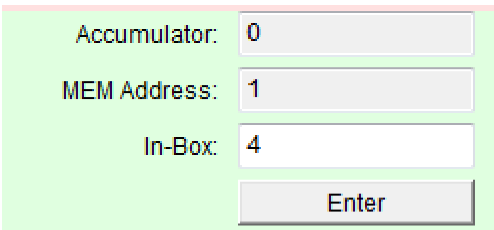
You can see that memory location 0 now contains 901 which is the instruction (opcode) for INP. Location 1 contains 902 which is OUT. Location 2 contains 0 which is HLT.

Running the program

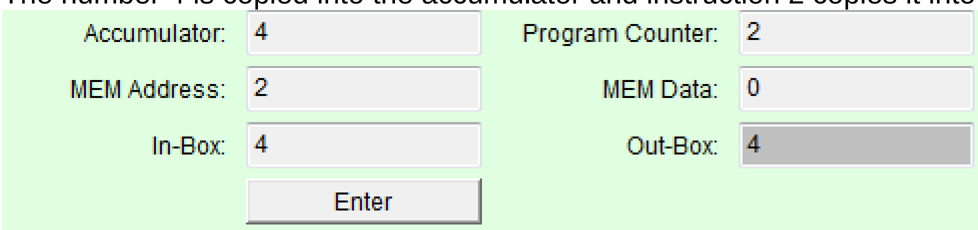
When the run button is clicked, the program counter is set to 1 and you can see that input is required.



We shall enter the number 4.



The number 4 is copied into the accumulator and instruction 2 copies it into the out-box.



The third instruction is HLT so the program ends.

PC = 2 : Instruction in Memory 2 is 0
--> 0 represents: HALT
--> Execution Stopped
Processor Stopped

Program 2

INP
STA FIRST
INP
ADD FIRST
OUT
INP
SUB FIRST
OUT
HLT
FIRST DAT

This program adds and subtracts numbers. Here it is in the message box.

Little Man Computer Memory:

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0
10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	0	0	0	0	0
20	21	22	23	24	25	26	27	28	29
0	0	0	0	0	0	0	0	0	0

Message Box:

INP
STA FIRST
INP
ADD FIRST
OUT
INP
SUB FIRST
OUT
HLT
FIRST DAT

When it is compiled, you can see the machine code in the memory locations.

Little Man Computer Memory:

0	1	2	3	4	5	6	7	8	9
901	309	901	109	902	901	209	902	0	0
10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	0	0	0	0	0
20	21	22	23	24	25	26	27	28	29
0	0	0	0	0	0	0	0	0	0
30	31	32	33	34	35	36	37	38	39
0	0	0	0	0	0	0	0	0	0
40	41	42	43	44	45	46	47	48	49
0	0	0	0	0	0	0	0	0	0
50	51	52	53	54	55	56	57	58	59
0	0	0	0	0	0	0	0	0	0
60	61	62	63	64	65	66	67	68	69
0	0	0	0	0	0	0	0	0	0
70	71	72	73	74	75	76	77	78	79

Message Box:

9 : FIRST DAT

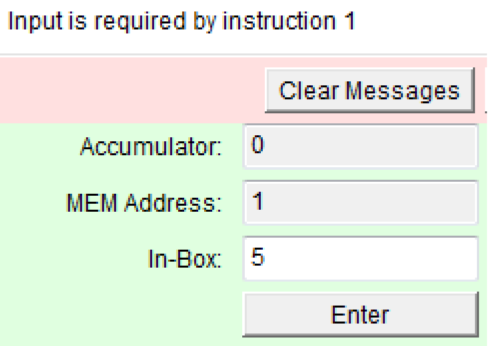
----- Resolving Labels -----
FIRST is a label for Address : 9

----- Translating Mnemonics -----
Line 0 : INP
    Opcode = 901
Line 1 : STA
    Opcode = 3 Address = 09
Line 2 : INP
    Opcode = 901
Line 3 : ADD
    Opcode = 1 Address = 09
Line 4 : OUT
    Opcode = 902
Line 5 : INP
    Opcode = 901
Line 6 : SUB
    Opcode = 2 Address = 09
Line 7 : OUT
    Opcode = 902
Line 8 : HLT
    Opcode = 0
Line 9 : DAT
----- Program Successfully Compiled -----

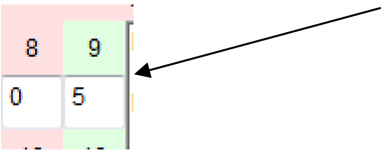
Address 0 contains 901 which means INP. It is asking for a number to be input, which is then stored in the accumulator.
Address 1 contains 309. This means store what is in the accumulator in address labelled FIRST, which is in fact address 9. Memory address 9 has been chosen by the assembler as being where the data labelled FIRST is to be stored.
Address 2 contains 901, asking for another input. This will go into the accumulator.
Address 3 contains 109. 1 is for ADD and 09 is address 9, so it means add to the accumulator whatever is in address 9, which is of course where the data labelled FIRST is found.
Address 4 contains 902 which is OUT. So the result of the addition, the number now in the accumulator is output.
Address 5 contains 901, asking for another input.
Address 6 contains 209 which means 2 for SUB and 09 means subtract from the accumulator whatever is in FIRST, i.e. address 9.
Address 7 contains another 902 which means output the contents of the accumulator.
Address 8 contains 0 which is HLT or halt the program.

We will now run the program with the values 5, 10 and 12. We expect the first output to be 15 because of the addition of 5 and 10. The second output will be 7 because 5 will be subtracted from the number 12 that we input.

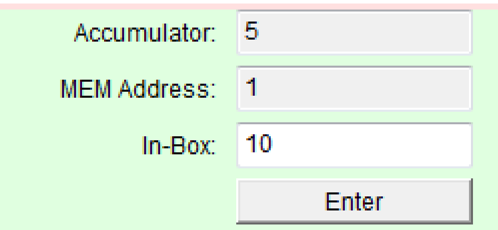
Here is 5 as the first input.



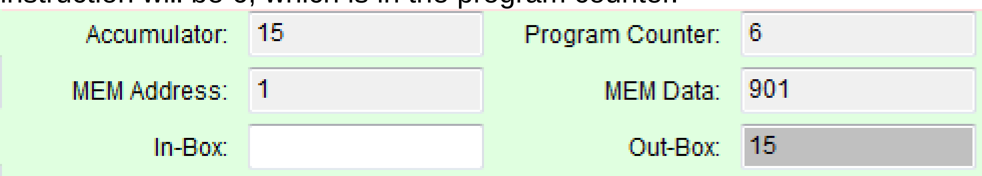
We can see that it has been stored in location 9 which is labelled FIRST.



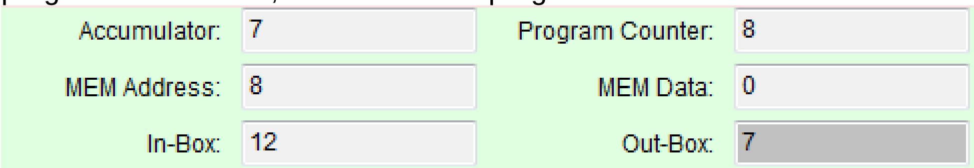
We enter 10.



We can see that the accumulator has taken the value 15 and also it has been output. You can see in the MEM data register that the computer is currently holding the OUT instruction, 901. The next instruction will be 6, which is in the program counter.



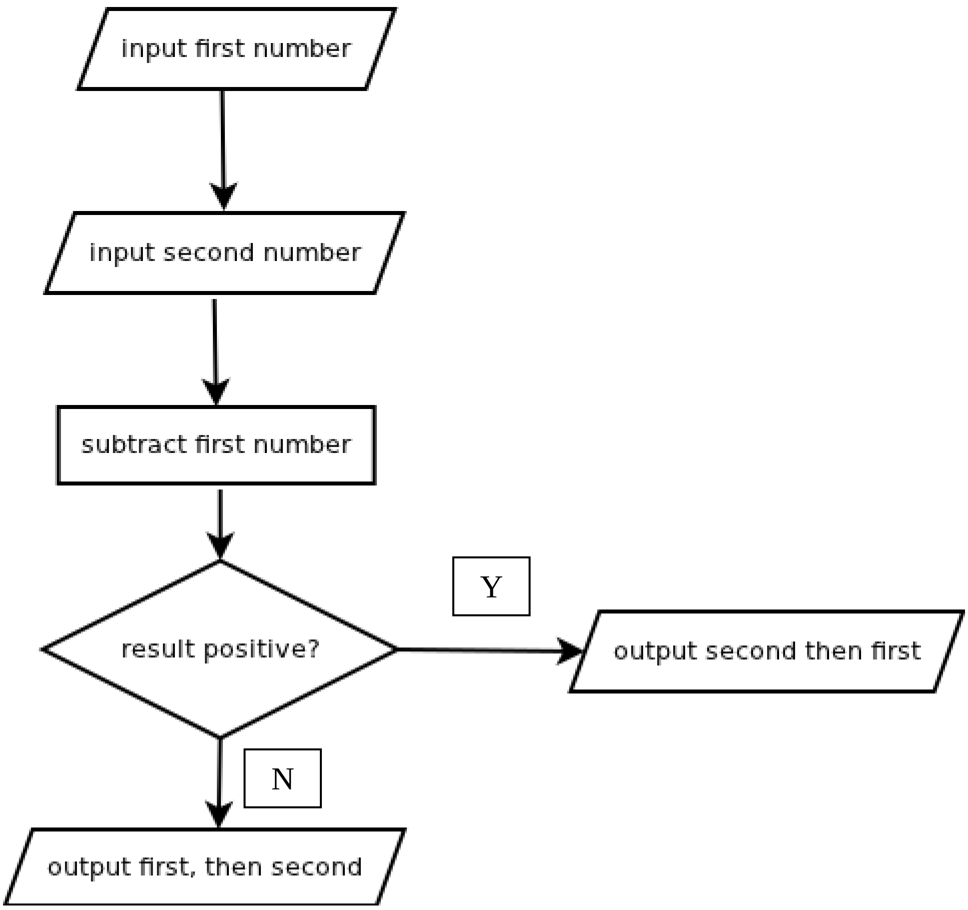
The instruction in 6 is again INP. We now enter 12 and click Enter. This puts 12 into the accumulator. Instruction 7 is immediately followed which is SUB FIRST. This subtracts 5, which is stored in the location labelled FIRST from the accumulator. We now have 7 in the accumulator. Instruction 7 is OUT, so the 7 is copied into the outbox. The next and final instruction in the program counter is 8, which halts the program.



Task 4. Write programs to run in LMC:

i. Take in two numbers and output the smaller first, then the larger.

Produce evidence to show that you have planned, written and tested your code.

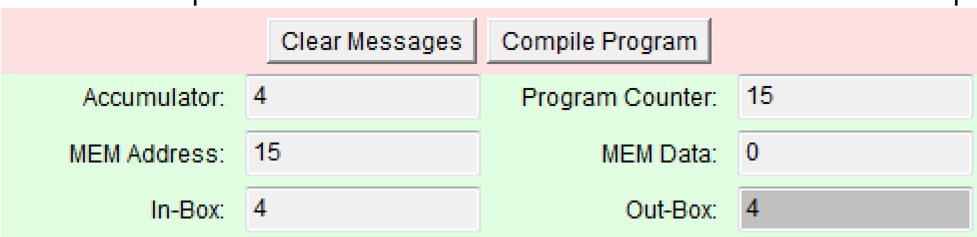


Here is the LMC code that achieves this.

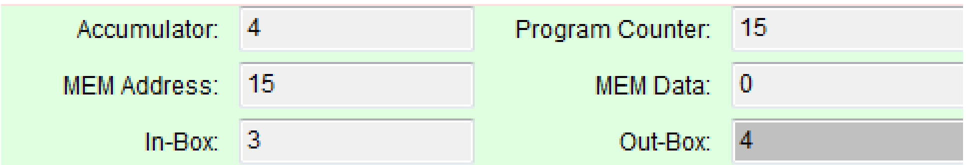
```
INP
STA FIRST
INP
STA SECOND
```

```
SUB FIRST
BRP SECONDBIG
LDA SECOND
OUT
LDA FIRST
OUT
BRA PROGEND
SECONDBIG LDA FIRST
OUT
LDA SECOND
OUT
PROGEND HLT
FIRST DAT
SECOND DAT
```

The output happens quickly so it is best to run it using the slow button.
Here is the output when tested with 3 then 4. The number 4 is the last output.



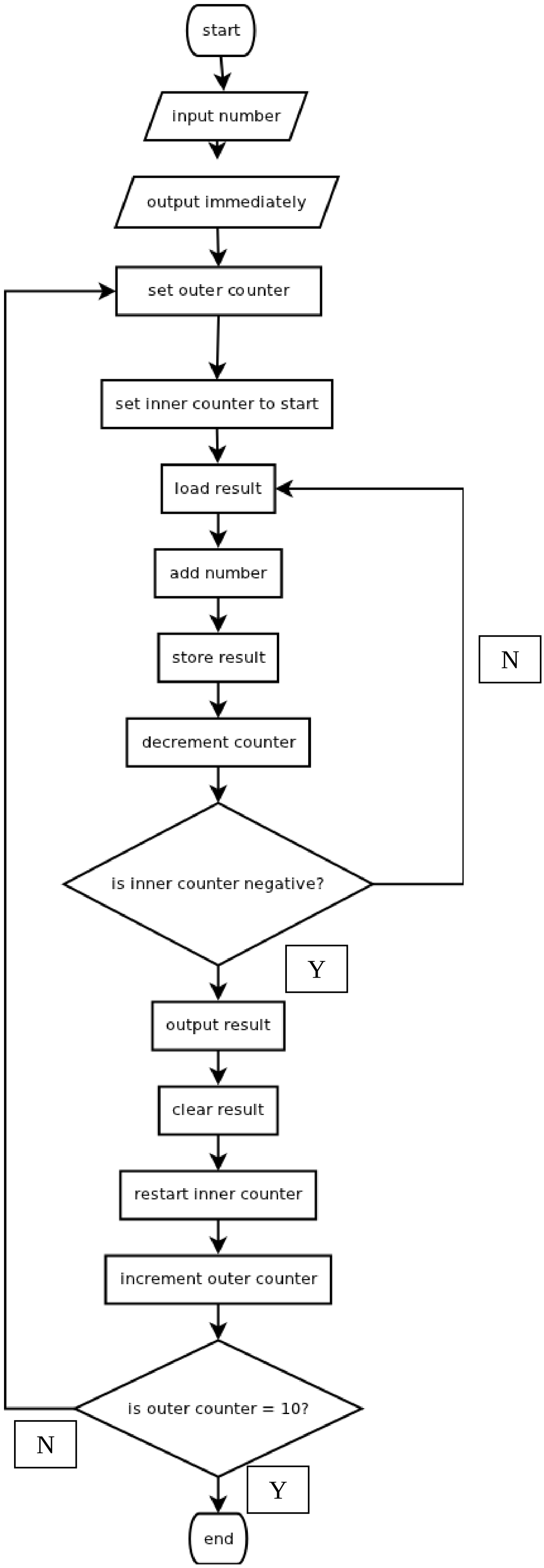
Here we test it with 4 then 3. Again 4 is the last output with 3 being visible as the last input into the in-box.



ii. Produce a multiplication table from 1 to 10 for any number input by the user.

Produce evidence to show that you have planned, written and tested your code.

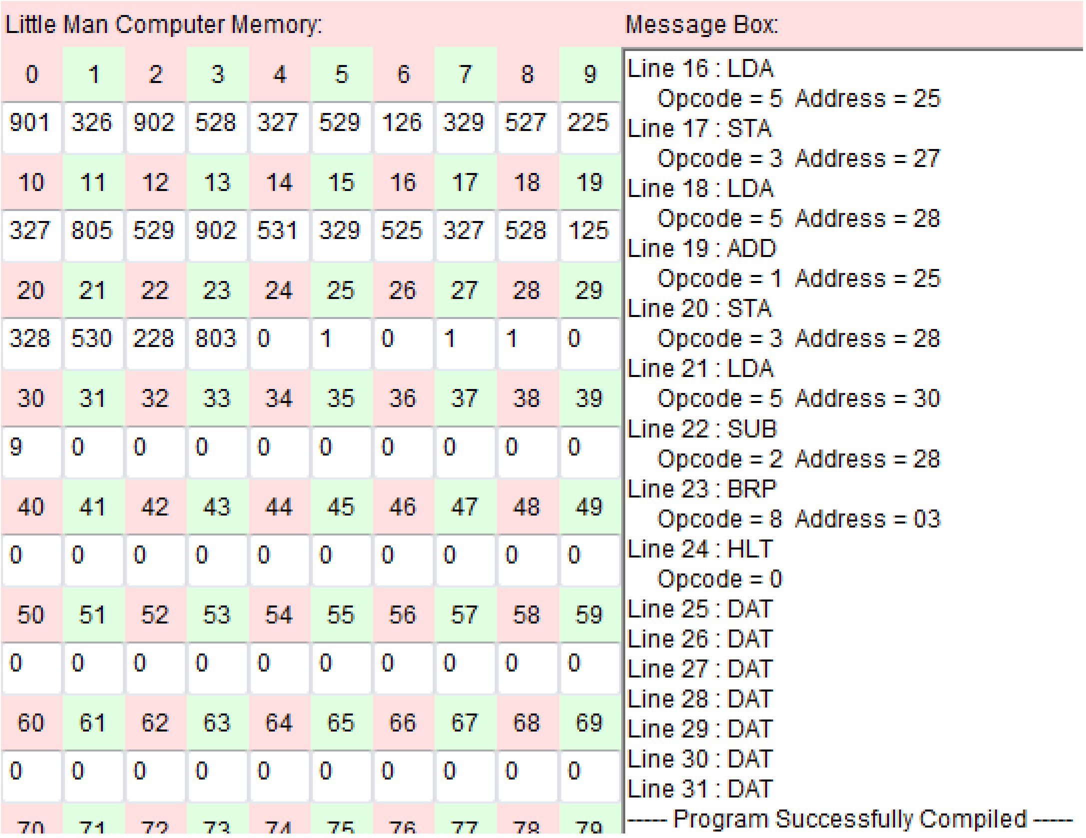
This requires multiplication to be done using multiple addition. The program asks for a number and then adds it to itself, the it adds it to itself again, outputting each result as it goes and so on until it has done this for each number up to ten. This is controlled by a loop, which terminates when the result of the countdown is negative. To make this happen ten times, the initial number is first output and then the addition process is made to happen ten times by having an outer loop controlled by another counter which increments each time a number has been output. When the counter produces a negative number when subtracted from 9, the loop terminates because all ten products have been output.


start
input number
output immediately
set outer counter
set inner counter to start
load result
add number
store result
decrement counter
N
is inner counter negative?
Y
output result
clear result
restart inner counter
increment outer counter
is outer counter = 10?
N
Y
end

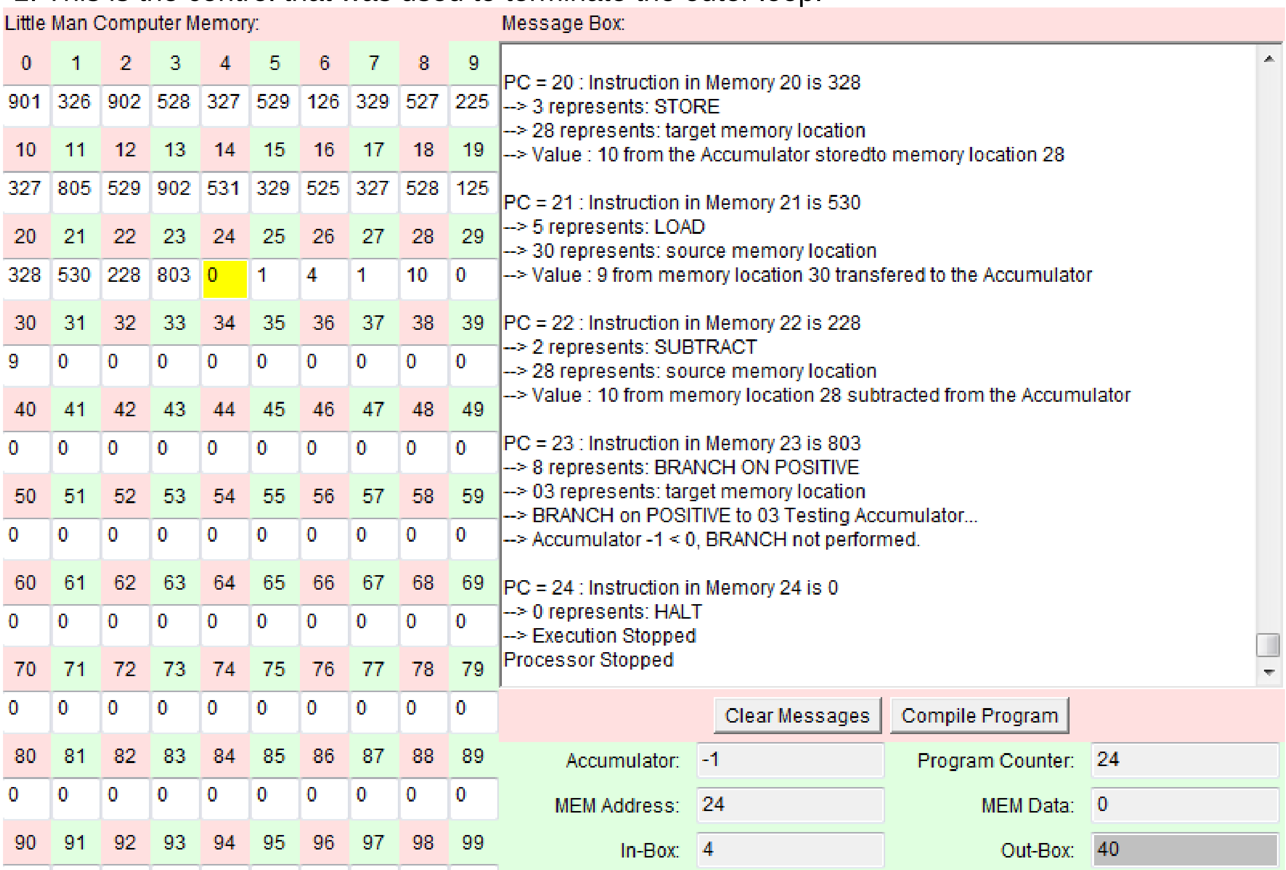
Here is the LMC code that makes this work.

```
INP
STA NUMBER1
OUT

OUTERLOOP LDA COUNTER
STA NUMBER2

INNERLOOP LDA RESULT
ADD NUMBER1
STA RESULT
LDA NUMBER2
SUB ONE
STA NUMBER2
BRP INNERLOOP

LDA RESULT
OUT

LDA ZERO
STA RESULT
LDA ONE
STA NUMBER2
LDA COUNTER
ADD ONE
STA COUNTER
LDA NINE
SUB COUNTER
BRP OUTERLOOP

HLT
ONE DAT 001
NUMBER1 DAT 000
NUMBER2 DAT 001
COUNTER DAT 001
RESULT DAT 000
NINE DAT 009
ZERO DAT 000
```

Here is the LMC with this code successfully compiled.



We shall test this with 4 as an input. We expect to get the 4 times table, 4,8,12,16 etc up to 40. The correct values appeared one after the other in the out-box. Here is the end condition showing the 4 we originally input and the 40 as the final product. You can see that the accumulator is set to -1. This is the control that was used to terminate the outer loop.



iii. Input five numbers and output them in reverse order.

Produce evidence to show that you have planned, written and tested your code.

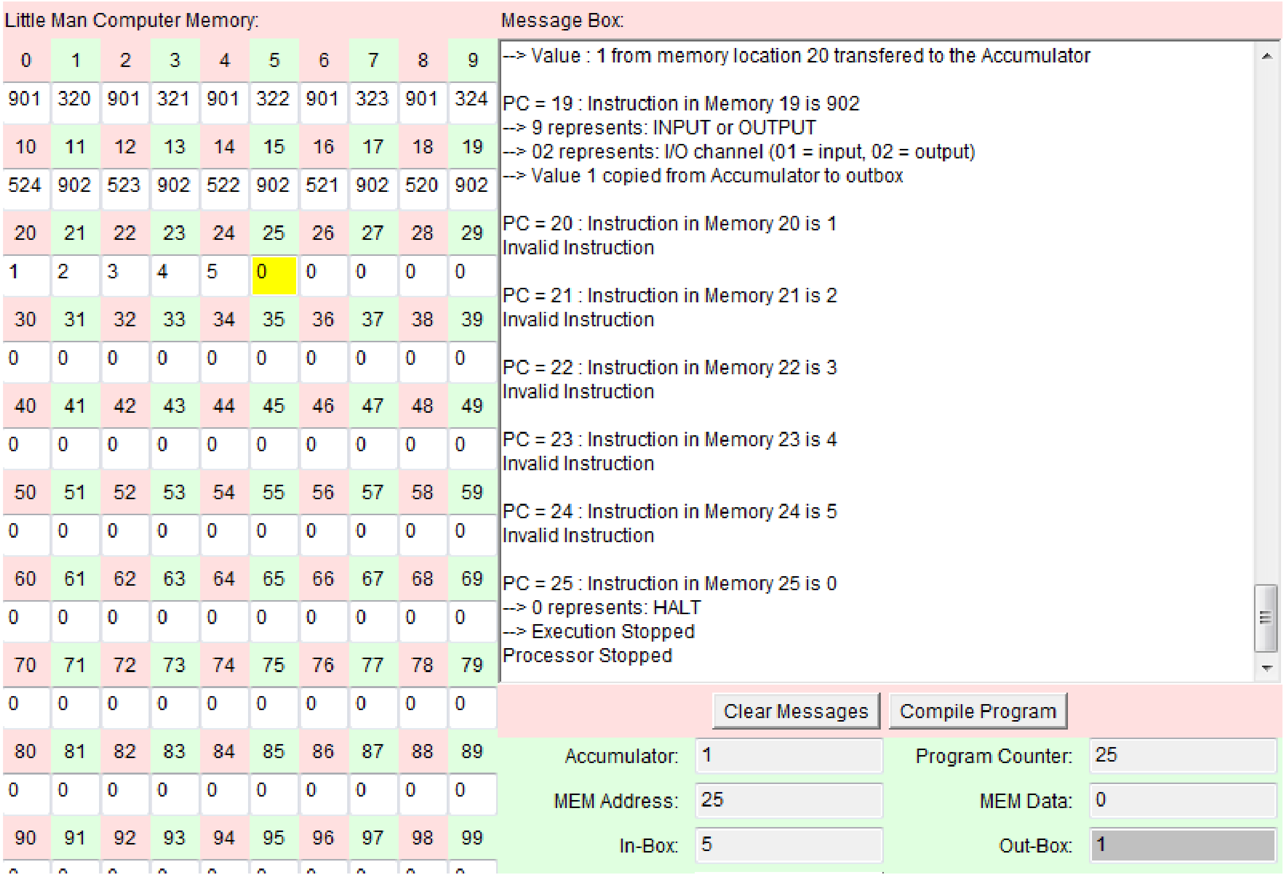
This can be done quite simply, by setting up five storage locations to accept the five numbers. The numbers can then be called back in any order the programmer wants. The use of labels makes this easy.

This is straightforward so it does not need a flow chart to illustrate it.

Here is the LMC code.

```
INP
STA ONE
INP
STA TWO
INP
STA THREE
INP
STA FOUR
INP
STA FIVE

LDA FIVE
OUT
LDA FOUR
OUT
LDA THREE
OUT
LDA TWO
OUT
LDA ONE
OUT

ONE DAT
TWO DAT
THREE DAT
FOUR DAT
FIVE DAT
```

Here is the final output from running this code. The inputs were 1,2,3,4,5. The LMC shows, at the end, the final output of 1 and the final input of 5. The outputs were 5,4,3,2,1 as expected.



Task 5. Produce an evaluation of your solutions.

The solutions all work perfectly and the code is mostly efficient. The multiplication uses a nested loop to produce the output with minimal code. The reversing of the numbers could have been made more elegant with loops, but with so few operations, the savings would have been minimal if at all.

Task 6. Write a conclusion about the possibility of writing effective and complex programs with only a limited instruction set.

This depends on how limited the set is. A rich instruction set makes programming easier because there is an instruction to do most of the things that you require. With a smaller set, you have to group instructions together in order to carry out the most simple task. In the case of the LMC, it is not possible to multiply or divide using a direct instruction, so multiplication has to be achieved by adding numbers together multiple times and division by multiple subtractions. So, a lot is possible with a small instruction set, but it leads to much harder work and the likelihood of errors. Also, solutions that are produced by long sections of code are likely to be slower when executed than solutions implemented in hardware.

The hard work involved in writing a program in LMC can be shown by looking at a simple program written in VB. This one

```
Sub Main()

        Dim totalnum As Integer
        Dim multiplier As Integer

        multiplier = 3

        For num = 1 To 10

            Console.WriteLine(multiplier * num)

        Next num

        Console.ReadKey()

    End Sub
```

In just a few lines, we have something that would take many lines in LMC.

The LMC has further big limitations in that it cannot handle characters. So it can do a lot of quite sophisticated mathematical operations but it cannot do anything with text, which makes it useless for many important computing applications.

The LMC has no features for acting directly on the hardware like most larger instruction sets. It also doesn’t have bitwise operations which reduces the variety of processes that can be carried out on data.

Candidates should complete all tasks.

The tasks are set so as to enable all the techniques identified in the specification to be demonstrated in their solution. The tasks provide opportunities to demonstrate a range of skills and all three tasks contribute to the overall mark awarded for this assessment. Marks are awarded for using the appropriate skills and techniques effectively and efficiently to produce a solution to these three tasks. Not all techniques will be required for each of the subtasks. You are required to identify the requirements for each task, design a solution using appropriate techniques, code the solution and test/evaluate this solution against the identified design criteria.

Task 1 Animal ages.

Design code and test a program to convert dog or cat years into their human equivalents. The program needs to ask the user for their choice of animal and should allow them to enter the age. The output should be the equivalent human age for the animal.

The formulae for converting these animal ages to human equivalents are:

DOG:

11 dog years per human year for the first 2 years, then 4 dog years per human year for each year after.

CAT:

15 years for the first year of life, 10 for the second year and 4 for each year after.

Task 2 System password.

Design, code test and evaluate a system to accept and test a password for certain characteristics.

It should be at least 6, and no more than 12 characters long

The system must indicate that the password has failed and why, asking the user to re-enter their choice until a successful password is entered.

A message to indicate that the password is acceptable must be displayed.

Password strength can be assessed against simple criteria to assess its suitability; for example a password system using only upper and lower case alphabetical characters and numeric characters could assess the password strength as:

WEAK if only one type used, e.g. all lower case or all numeric

MEDIUM if two types are used

STRONG if all three types are used.

For example

hilltop, 123471324, HAHGFD are all WEAK,

catman3 and 123456t are MEDIUM and

RTH34gd is STRONG

A message to indicate the password strength should be displayed after an acceptable password is chosen.

Task 3 High scores database. 15 marks

Design, code and test a system to store and manage user names and their highest score.

The system must be able to

create a file

add data to a file

locate data in the file by name and their highest score

delete an item and its associated data from the file

locate and update a high score for a user

GCSE Computing Controlled Assessment

Unit A453 Coding a solution

Unit Recording Sheet

OCR RECOGNISING ACHIEVEMENT

Please read the instructions printed on the other side of this form. **One** of these Unit Recording Sheets, suitably completed, should be attached to the assessed work of **each** candidate.

Unit	**A453**		**Year**	**201**
Centre Name			**Centre Number**	
Candidate Name			**Candidate Number**	

	Guidance			**Teacher Comment**	**Mark**
Use of programming techniques	There is an attempt to solve parts of the tasks using few of the techniques identified. **[0 - 2]**	There is an attempt at most parts of the tasks using several techniques. **[3 - 4]**	There is an attempt to solve all of the tasks using most of the techniques listed. **[5 - 6]**	All three tasks have been attempted though not all of task 3 was completed. Whilst not all techniques have been used a good number have.	5 **Max 6**
Efficient use of programming techniques	The techniques used may not be entirely appropriate to the problem and will only produce partially working solutions to a small part of the problem. **[0 - 4]**	The techniques will be used appropriately giving working solutions to most of the parts of the problem. Some sections of the solution will be inefficiently coded. **[5 - 8]**	The techniques are used appropriately in all cases giving an efficient, working solution for all parts of the problem. **[9 - 12]**	The techniques are generally used appropriately but coding does often lack efficiency. Most parts of the problems have been completed but not all (e.g. task 3 delete).	7 **Max 12**

Design	There will be vague comments on what the task involves and a vague outline describing the intended approach to some parts of the problem. There will be brief comments on how this might be tested but with no mention of success criteria. **[0 - 3]**	There will be a brief analysis of the tasks indicating what is required for each of the tasks. There will be a set of basic algorithms outlining a solution to most parts of the problem. There will be some discussion of how this will be tested and how this compares to the identified outcomes in the tasks. There will be discussion of the variables to be used and some general discussion of validation **[4 - 6]**	There will be a detailed analysis of what is required for these tasks justifying their approach to the solution. There will be a full set of detailed algorithms representing a solution to each part of the problem. There will be detailed discussion of testing and success criteria. The variables and structures will be identified together with any validation required. **[7 - 9]**	Brief outline of what task involves. The algorithms vary in terms of detail. There is very little evidence of testing being planned aside from a cursory attempt in Ex1.	4 **Max 9**
Development	There will be some evidence to show a solution to part of the problem with some evidence to show that it works. Code will be presented with little or no annotation, the variable names not reflecting their purpose and with little organisation or structure. **[0 - 3]**	There will be evidence to show how the solutions were developed. There will be some evidence of testing during development showing that many parts of the solution work. The code will be organised with sensible variable names and with some annotation indicating what sections of the code does. **[4 – 6]**	There will be detailed evidence showing development of the solution with evidence of systematic testing during development to show that all parts work as required. The code will be well organised with meaningful variable names and detailed annotation indicating the function of each section. **[7- 9]**	Code is lacking comments. Variable names are sometimes cryptic or ambiguous (e.g. the use of both pwd$ and pass$ in Exercise 2)	4 **Max 9**

Testing	There will be evidence to show that the system has been tested for function but the test plan will be limited in scope with little structure. There will be little or no evidence to show how the result matches the original criteria. The evidence of written communication is limited with little or no use of specialist terms. Errors in spelling, punctuation and grammar may be intrusive. Information may be ambiguous or disorganised. There will be some comments on others' and their own input into group work. **[0 - 3]**	There will be a test plan covering many parts of the problem with some suggested test data. There will be evidence that the system has been tested using this data. There will be some evidence to show how the results of testing have been compared to the original criteria. There will be a brief discussion of how successful or otherwise the solutions are. Produces evidence of good written communication using some specialist terms. There will be few errors in spelling, grammar and punctuation. Information for the most part will be presented in a structured format. They will have commented on their own and others' contribution to any group work and **[4 - 6]**	The test plan will cover all major success criteria for the original problem with evidence to show how each of these criteria have been met, or if they have not been met, how the issue might be resolved. There will be a full evaluation of the final solution against the success criteria. A high level of written communication will be obvious throughout the task and specialist terms/technology with accurate use of spelling will have been used. Grammar and punctuation will be used correctly and information will be presented in a coherent and structured format. They will provide an evaluation on theirs and others' contribution to any group activities. **[7 - 9]**	Testing has taken place but is far from exhaustive. There is some evidence of testing and programs working. There are a few SPaG errors and these are not intrusive Evaluation states whether tasks were successful but lacks any analysis.	4 **Max 9**
				Total/45	**0** 24

Guidance on Completion of this Form

1. **One** sheet should be used for each candidate.
2. Please ensure that the appropriate boxes at the top of the form are completed.
3. Using the guidance identify the most appropriate mark range for the work and enter the mark awarded for each element in the mark column.
4. Add appropriate comments to assist the moderator in the 'Teacher Comment' column.
5. Add the marks for the strands together to give a total out of 45. Enter this total in the relevant box.

A453

Task 1: Animal Age

Analysis

A system to convert cat or dog ages into human equivalents

Rules

DOG:

11 years per year for 2 years then 4 per extra year

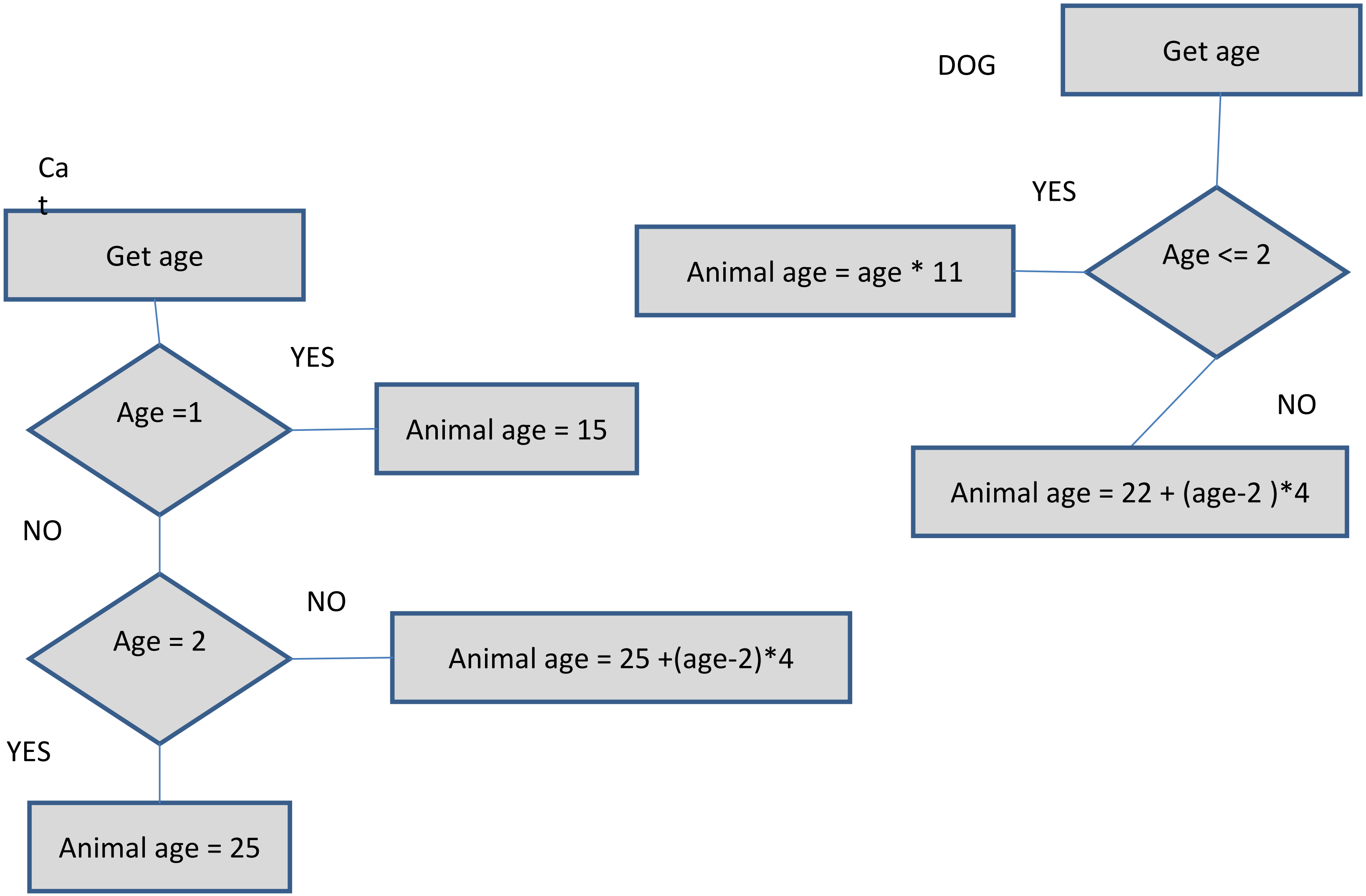
CAT

15 years for first year, 10 for second then 4 per year

Analysis

- Need to get choice of cat or dog
- Need to get animal age in whole years
- Need to convert animal age to human equivalent using rules and print result.

Design


DOG
Get age
YES
Age <= 2
Animal age = age * 11
NO
Animal age = 22 + (age-2)*4
Ca
t
Get age
YES
Age =1
Animal age = 15
NO
NO
Age = 2
Animal age = 25 +(age-2)*4
YES
Animal age = 25

Development and testing

The code ask the user to input either dog or cat
If dog checks for age <=2
If <= 2 then age*2
If not then 22+ (age-2)*4

If cat checks
if 1 year then age 15
If 2 years the age 25
If not then 25+(age-2)*4

```
PRINT "Cat and Dog age to human equivalent"
  REPEAT
   INPUT "Choose Cat or Dog, type cat or dog", a$
   INPUT "enter the age in years"; a
   IF a$="dog" THEN
    IF a<=2 THEN
     animalage = e*11
    ELSE
     animalage = 22+(a-2)*4
    ENDIF
   ENDIF

   IF a$="cat" THEN
    IF a <= 1 THEN
     animalage =a* 15
    ELSE
     IF a <= 2 THEN
      animalage =15 +(a-1)*10
     ELSE
      animalage= 25+(a-2)*4
     ENDIF
    ENDIF
   ENDIF

   PRINT "Your animal's age in human years is", animalage
```

Development and testing

	Test	Data	Expected
2	dog ages	1	11
3		2	22
5	Cat ages	1	15
6		2	25
7		3	29

Evidence

```
dogD
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? dog
enter the age in years? 1
Your animal's age in human years is            11
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? dog
enter the age in years? 2
Your animal's age in human years is            22
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? cat
enter the age in years? 1
Your animal's age in human years is            15
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? cat
enter the age in years? 2
Your animal's age in human years is            25
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? cat
enter the age in years? 3
Your animal's age in human years is            29
>_
```

Evaluation:

I enjoyed doing this task. It was quite easy and it works as I expected.

A453

Task 2: Password strength sample task

Analysis

1. Input a password
2. Is the password between 6 and 12 characters long?

   No; reject and return to stage 1

   YES; output message and carry on
3. Check each character of the password in turn

   Is this character upper case? If yes flag that upper case is included

   Is this character lower case? If yes flag that lower case is included

   Is this character a number? If yes flag that number is included
4. If three flags set then the password is STRONG
5. If two flags set then the passwords is MEDIUM
6. If one flag set then the password is WEAK

Design

Check password length less than 6 error go back
Check password length greater than 12 error go back
Password OK
Check each character

  between a and z, set low to 1
  between A and Z set upp to 1
  between 0 and 9 set num to 1

Add upp, low and num to get password strength
1 weak
2 medium
3 strong

Development & Testing


```
pwd$="NOTOK"
REPEAT
  INPUT "Password" pass$
  IF LEN(pass$)< 6 THEN
    PRINT "too short"
  ELSE
    IF LEN(pass$)>12 THEN
      PRINT"too long"
    ELSE
      PRINT"OK"
      pwd$="OK"
    ENDIF
  ENDIF
UNTIL pwd$="OK"
```

Check 6 to 12 characters in password. It works.

Development & Testing


```
upp=0
  low=0
  num=0
  FOR x=1 TO LEN(pass$)
   IF MID$(pass$,x,1) >="a" AND MID$(pass$,x,1)<="z" THEN
    low=1
   ENDIF
   IF MID$(pass$,x,1) >="A" AND MID$(pass$,x,1)<="Z" THEN
    upp=1
   ENDIF
   IF MID$(pass$,x,1) >="0" AND MID$(pass$,x,1)<="9" THEN
    num=1
   ENDIF
  NEXT
  str=upp+low+num
  IF str=1 THEN
   PRINT "Weak"
  ELSE
   IF str=2 THEN
    PRINT "medium"
   ELSE
    IF str=3 THEN
     PRINT "strong"
    ENDIF
   ENDIF
  ENDIF
```

Checking mix of upper, lower and number gives right result. It works

Development & Testing

Completed code

Checks each character in the password in turn to see if it is between a and z then A and Z then 0 and 9. It sets low, upp and num to one if it finds one of them and adds them up to get the overall strength.

```
pwd$="NOTOK"
REPEAT
  INPUT "Password" pass$
  IF LEN(pass$)< 6 THEN
    PRINT "too short"
  ELSE
    IF LEN(pass$)>12 THEN
      PRINT"too long"
    ELSE
      PRINT"OK"
      pwd$="OK"
    ENDIF
  ENDIF
UNTIL pwd$="OK"


upp=0
low=0
num=0
FOR x=1 TO LEN(pass$)
  IF MID$(pass$,x,1) >="a" AND MID$(pass$,x,1)<="z" THEN
    low=1
  ENDIF
  IF MID$(pass$,x,1) >="A" AND MID$(pass$,x,1)<="Z" THEN
    upp=1
  ENDIF
  IF MID$(pass$,x,1) >="0" AND MID$(pass$,x,1)<="9" THEN
    num=1
  ENDIF
NEXT
str=upp+low+num
IF str=1 THEN
  PRINT "Weak"
ELSE
  IF str=2 THEN
    PRINT "medium"
  ELSE
    IF str=3 THEN
      PRINT "strong"
    ENDIF
  ENDIF
ENDIF
```

Checking length OK

F1 for Help 550 31,43 NUM

Evaluation of the solution

The program was tested with passwords with less than 6, more than 12 and it asked for the password again

The program was tested with mixed passwords using numbers. Upper case and lower case, it correctly got the right strength each time.

A453

Task 3: High scores table sample task

Analysis

- A system to manage high scores
  - Create a file and be able to
  - Find a score for a user
  - Update a score for a user
  - Add a new user and score
  - Delete a user and score

Analysis

- Need to check if the file exists, if not create one
- Need to load data from the file into an array
- Need to check if username exists to update score, if not error message
- Need an option system for edit, new and delete, otherwise error message
- Need to write changed data back to file

Design

Use routine to check if file exists and if not create it, otherwise open it.
Read data into arrays for names and scores
Get option edit, new or delete
If edit search for name
Get new score in array
Write data back to file
If new
Get data for next array items
Write data to file
Delete
Find data
Delete from array
Write data to file

Design / development

Check file exists: if I try to open a file it returns 0 if the file doesn't exist so I can use this to decide if I need to create a file. I will use this code I found on a website and changed.

If openin file = 0 the create file else open file

```
IF OPENIN "c:\users\george\scores.txt" =0 THEN
    chan1 = OPENOUT "c:\users\george\scores.txt"
    CLOSE#chan1
   ENDIF
```

Design / development

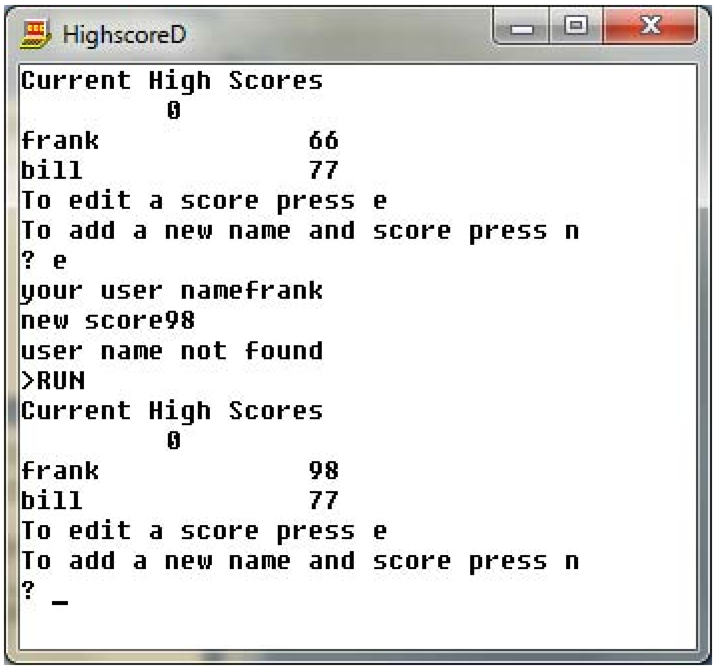
I need to create an array for the names and scores so that I can read in the data from a file.

I have created a simple text file with some names and scores to test this section of code.

```
DIM name$(10)
DIM score(10)

    x=1
    IF OPENIN "c:\users\george\scores.txtt" =0 THEN
      chan1 = OPENOUT "c:\users\george\scores.txt"
      CLOSE#chan1
    ENDIF

    chan1=OPENIN "c:\users\george\scores.txt"
    REPEAT
      INPUT#chan1,name$(x)
      INPUT#chan1,score(x)
      ix=x+1
    UNTIL EOF#chan1
    CLOSE#chan1
     x=x-1
```

Design / development

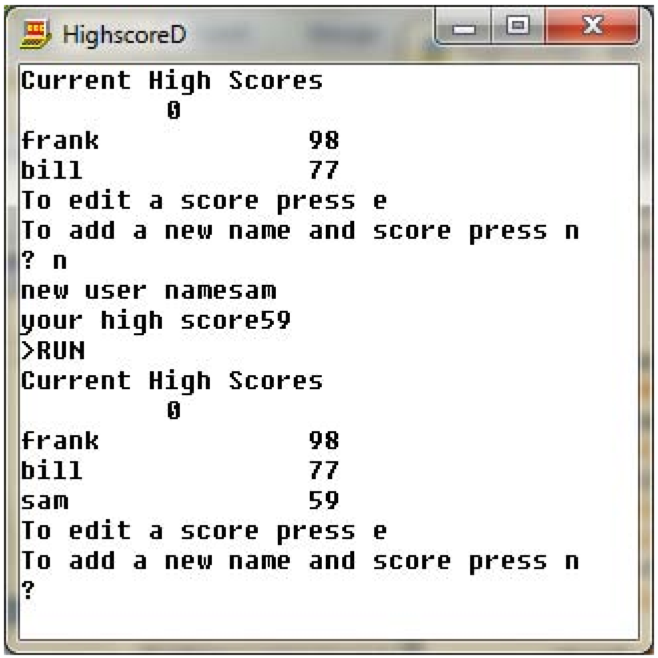
The edit routine should search for the user name in the array, edit the score and write the new score to the array. If not found it should print and error message.




```
PRINT "To edit a score press e"
  PRINT "To add a new name and score press n"
  INPUT select$
  IF select$="e"  THEN
   INPUT "your user name" user$
   c=1
   WHILE  c<=x
    IF user$= name$(c) THEN
     INPUT "new score" newscore
     score(c) = newscore
    ENDIF
    IF c>=x THEN
     PRINT "user name not found"
    ENDIF
    c=c+1
   ENDWHILE

   chan2=OPENOUT "c:\users\george\scores.txt"
   FOR c=1 TO x
    PRINT#chan2,name$(c),score(c)
   NEXT c
   CLOSE#chan2
```

Works; frank has been updated from 66 to 98

Design / development



The data for sam has been added at the end of the array as expected.

To add the new data feature If n is pressed it starts this section of code:

```
 IF select$="n" THEN

     INPUT "new user name" newname$
     INPUT "your high score" highscore
     x=x+1
     name$(x)= newname$
     score(x)= highscore

     chan2=OPENOUT "c:\users\george\scores.txt"
     FOR c=1 TO x
       PRINT#chan2,name$(c),score(c)
     NEXT c
     CLOSE#chan2
   ENDIF
```

Testing and evaluation

I tested the program as I wrote it and the evidence is in the development.
This was not as easy as the first tasks and I was not able to complete the delete option.

The error message user not found keeps printing on screen and I think I can fix this by using an IF to check if a change has been made before printing this message.

```
DIM name$(10)
  DIM score(10)

  x=1
  IF OPENIN "c:\users\george\scores.txt" =0 THEN
   chan1 = OPENOUT "c:\users\george\scores.txt"
   CLOSE#chan1
  ENDIF

  chan1=OPENIN "c:\users\george\scores.txt"
  REPEAT
   INPUT#chan1,name$(x)
   INPUT#chan1,score(x)
   x=x+1
  UNTIL EOF#chan1
  CLOSE#chan1
  x=x-1

  PRINT "Current High Scores"
  FOR i = 1 TO x
   PRINT name$(i),score(i)
  NEXT i
```

```
PRINT "To edit a score press e"
  PRINT "To add a new name and score press n"
  INPUT select$
  IF select$="e"  THEN
   INPUT "your user name" user$
   c=1
   WHILE  c<=x
    IF user$= name$(c) THEN
     INPUT "new score" newscore
     score(c) = newscore
    ENDIF
    IF c>=x THEN
     PRINT "user name not found"
    ENDIF
    c=c+1
   ENDWHILE

   chan2=OPENOUT "c:\users\george\scores.txt"
   FOR c=1 TO x
    PRINT#chan2,name$(c),score(c)
   NEXT c
   CLOSE#chan2
  ENDIF
```

```
  IF select$="n" THEN

  INPUT "new user name" newname$
  INPUT "your high score" highscore
  x=x+1
  name$(x)= newname$
  score(x)= highscore

  chan2=OPENOUT "c:\users\george\scores.txt"
  FOR c=1 TO x
   PRINT#chan2,name$(c),score(c)
  NEXT c
  CLOSE#chan2
 ENDIF
```

1258292505

GCSE Computing Controlled Assessment

Unit A453 Coding a solution

Unit Recording Sheet

OCR
RECOGNISING ACHIEVEMENT

Please read the instructions printed on the other side of this form. **One** of these Unit Recording Sheets, suitably completed, should be attached to the assessed work of **each** candidate.

Unit	**A453**		**Year**	**2**	**0**		

Centre Name		**Centre Number**					
Candidate Name		**Candidate Number**					

	Guidance			**Teacher Comment**	**Mark**
Use of programming techniques	There is an attempt to solve parts of the tasks using few of the techniques identified. **[0 - 2]**	There is an attempt at most parts of the tasks using several techniques. **[3 - 4]**	There is an attempt to solve all of the tasks using most of the techniques listed. **[5 - 6]**	An attempt has been made at all three tasks. A good range of techniques has been sensibly used.	6 **Max 6**

Efficient use of programming techniques	The techniques used may not be entirely appropriate to the problem and will only produce partially working solutions to a small part of the problem. **[0 - 4]**	The techniques will be used appropriately giving working solutions to most of the parts of the problem. Some sections of the solution will be inefficiently coded. **[5 - 8]**	The techniques are used appropriately in all cases giving an efficient, working solution for all parts of the problem. **[9 - 12]**	Techniques are used appropriately. Code is generally efficient. There some inefficiencies. For example the candidate has used 3 FOR loops in task 2 where one would suffice. Program 3, particularly, would benefit from some modularity.	9 **Max 12**
Design	There will be vague comments on what the task involves and a vague outline describing the intended approach to some parts of the problem. There will be brief comments on how this might be tested but with no mention of success criteria. **[0 - 3]**	There will be a brief analysis of the tasks indicating what is required for each of the tasks. There will be a set of basic algorithms outlining a solution to most parts of the problem. There will be some discussion of how this will be tested and how this compares to the identified outcomes in the tasks. There will be discussion of the variables to be used and some general discussion of validation **[4 - 6]**	There will be a detailed analysis of what is required for these tasks justifying their approach to the solution. There will be a full set of detailed algorithms representing a solution to each part of the problem. There will be detailed discussion of testing and success criteria. The variables and structures will be identified together with any validation required. **[7 - 9]**	Problems have been analysed. Algorithms are well designed using flow diagrams. Test Strategy/success criteria discussed for Ex 1+2 but not 3 No explicit discussion of variables or structures but validation is looked at.	8 **Max 9**

Development	There will be some evidence to show a solution to part of the problem with some evidence to show that it works. Code will be presented with little or no annotation, the variable names not reflecting their purpose and with little organisation or structure. **[0 - 3]**	There will be evidence to show how the solutions were developed. There will be some evidence of testing during development showing that many parts of the solution work. The code will be organised with sensible variable names and with some annotation indicating what sections of the code does. **[4 – 6]**	There will be detailed evidence showing development of the solution with evidence of systematic testing during development to show that all parts work as required. The code will be well organised with meaningful variable names and detailed annotation indicating the function of each section. **[7- 9]**	There is evidence of solution development and some testing during development. Meaningful variable names have been used. Occasionally code lacks structure (e.g. in the Delete section of Task 3 a FOR loop is overlapped by an IF rather than using nesting.) Commenting, when used is effective, but is too often missing.	7 **Max 9**

Testing	There will be evidence to show that the system has been tested for function but the test plan will be limited in scope with little structure. There will be little or no evidence to show how the result matches the original criteria. The evidence of written communication is limited with little or no use of specialist terms. Errors in spelling, punctuation and grammar may be intrusive. Information may be ambiguous or disorganised. There will be some comments on others' and their own input into group work. **[0 - 3]**	There will be a test plan covering many parts of the problem with some suggested test data. There will be evidence that the system has been tested using this data. There will be some evidence to show how the results of testing have been compared to the original criteria. There will be a brief discussion of how successful or otherwise the solutions are. Produces evidence of good written communication using some specialist terms. There will be few errors in spelling, grammar and punctuation. Information for the most part will be presented in a structured format. They will have commented on their own and others' contribution to any group work and **[4 - 6]**	The test plan will cover all major success criteria for the original problem with evidence to show how each of these criteria have been met, or if they have not been met, how the issue might be resolved. There will be a full evaluation of the final solution against the success criteria. A high level of written communication will be obvious throughout the task and specialist terms/technology with accurate use of spelling will have been used. Grammar and punctuation will be used correctly and information will be presented in a coherent and structured format. They will provide an evaluation on theirs and others' contribution to any group activities. **[7 - 9]**	Tests for Ex2+3 cover most eventualities and are backed up with evidence. Testing in Ex3 is lacking the same amount of rigour. Where issues have been encountered these have been discussed along with their resolution. Spelling punctuation and grammar are all of a good quality. None of the tasks required group work .	8 **Max 9**
				Total/45	38

Guidance on Completion of this Form

1 **One** sheet should be used for each candidate.
2 Please ensure that the appropriate boxes at the top of the form are completed.
3 Using the guidance identify the most appropriate mark range for the work and enter the mark awarded for each element in the mark column.
4 Add appropriate comments to assist the moderator in the 'Teacher Comment' column.
5 Add the marks for the strands together to give a total out of 45. Enter this total in the relevant box.

A453

Task 1: Animal Age

Analysis

A system to convert cat or dog ages into human equivalents

Rules

DOG:

11 years per year for 2 years then 4 per extra year

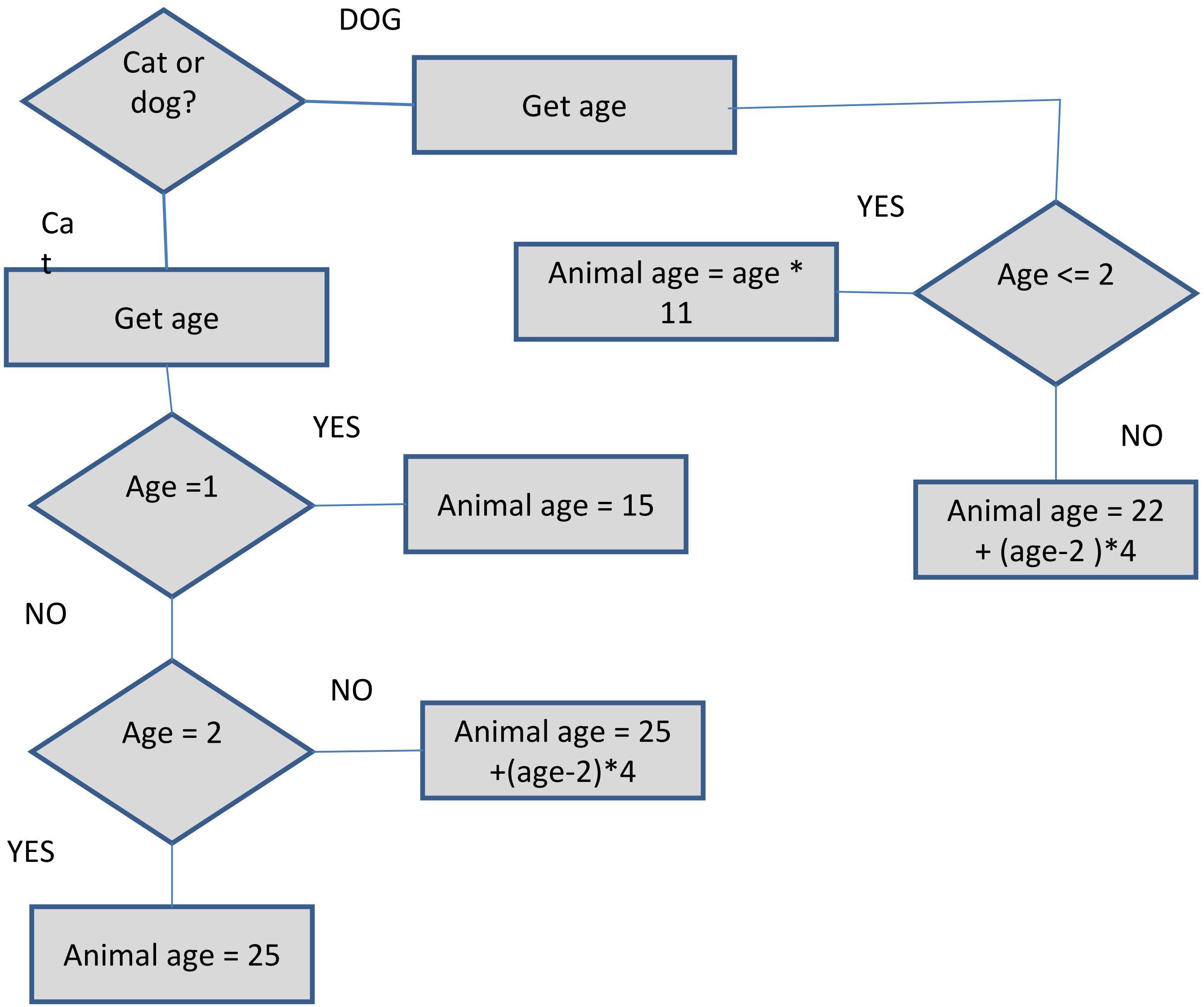
CAT

15 years for first year, 10 for second then 4 per year

Analysis

- Need to get choice of cat or dog
- Need to get animal age in whole years
- Need to convert animal age to human equivalent using rules and print result.

Design



Cat or dog?
DOG
Get age
Ca
t
Get age
YES
Animal age = age * 11
Age <= 2
YES
Age =1
Animal age = 15
NO
Animal age = 22 + (age-2)*4
NO
NO
Age = 2
Animal age = 25 +(age-2)*4
YES
Animal age = 25

Development and testing

The code loops to make sure user inputs either dog or cat
If dog checks for age <=2
If <= 2 then age*2
If not then 22+ (age-2)*4

If cat checks
if 1 year then age 15
If 2 years the age 25
If not then 25+(age-2)*4

Using print formatting to output answer as sentence with variables to complete the sentence.

```
PRINT "Cat and Dog age to human equivalent"
   REPEAT
    INPUT "Choose Cat or Dog, type cat or dog", animal$
   UNTIL animal$ ="cat" OR animal$="dog"
   PRINT "enter the ";animal$;"'s age in years";
   INPUTage

   IF animal$="dog" THEN
    IF age<=2 THEN
     animalage = age*11
    ELSE
     animalage = 22+(age-2)*4
    ENDIF
   ENDIF

   IF animal$="cat" THEN
    IF age = 1 THEN
     animalage =15
    ELSE
     IF age = 2 THEN
      animalage =25
     ELSE
      animalage= 25+(age-2)*4
     ENDIF
    ENDIF
   ENDIF

   PRINT "Your ";animal$;" is ";animalage;" years in human
terms"
```

Development and testing

	Test	Data	Expected
1	Check if dog or cat	cat, dog, rat, Dog, CAT	cat, dog accepted, rat, Dog and CAT clears and ask question again
2	dog ages	1,2	Expect 11 or 22
3		3,4	Expect 26, 30
4		3.5	Expect 28
5	Cat ages	1, 2	Expect 15, 25
6		3,4	Expect 29, 33
7		3.5	Expect 27
8	Cat and dog ages	Hat	Expect error program stops

Evidence for 1 and 2

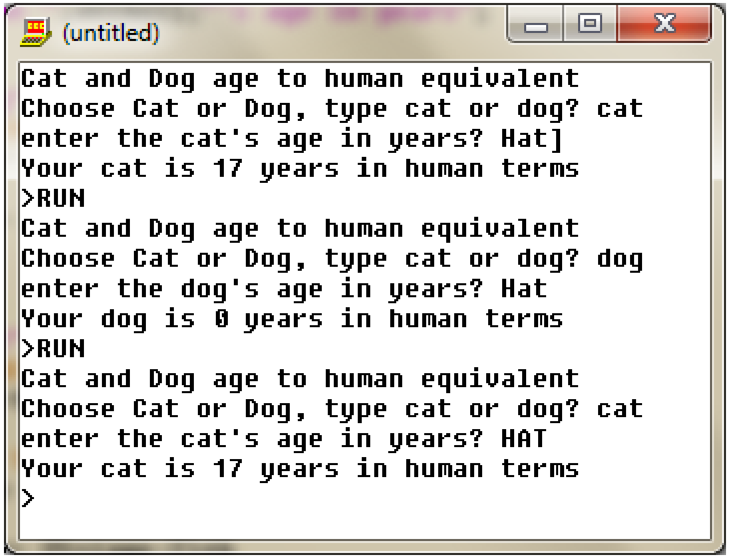
```
(untitled)
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? CAT
Choose Cat or Dog, type cat or dog? Dog
Choose Cat or Dog, type cat or dog? rat
Choose Cat or Dog, type cat or dog? dog
enter the dog's age in years? 1
Your dog is 11 years in human terms
>
```

Evidence for 2, 3,4

```
(untitled)
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? dog
enter the dog's age in years? 2
Your dog is 22 years in human terms
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? dog
enter the dog's age in years? 3
Your dog is 26 years in human terms
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? dog
enter the dog's age in years? 4
Your dog is 30 years in human terms
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? dog
enter the dog's age in years? 3.5
Your dog is 28 years in human terms
>_
```

Evidence for 1,5,6,7

```
(untitled)
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? cat
enter the cat's age in years? 1
Your cat is 15 years in human terms
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? cat
enter the cat's age in years? 2
Your cat is 25 years in human terms
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? cat
enter the cat's age in years? 3
Your cat is 29 years in human terms
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? cat
enter the cat's age in years? 4
Your cat is 33 years in human terms
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? cat
enter the cat's age in years? 3.5
Your cat is 31 years in human terms
>_
```



Test 7 NOT as expected. Why?

Cat makes sense, Hat taken as zero, therefore 25+ (0-2)*4 = 25-8 =17

Dog also makes sense hat taken as zero, so 0 is <=2, therefore age = 0*11

Fix:

Need to reject non numeric inputs though decimal ones appear to give a decent result.

Check 0.5 and 1.5 years for cat and dog but change age = 1 and age = 2 for cat to <=

Also only accept age >0

	Test	Data	Expect
8	Non numeric data	hat	Reject and ask for age again
9	Decimal values	Cat age 0.5	Expect 7.5
10	Decimals	Cat and dog 1.5	Expect 20 and 16.5
11	Negative values	Cat and dog -2, -3.98	Expect rejected

Evidence 8 and 9

(untitled)

```
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? cat
enter the cat's age in years? hat
? .5
Your cat is 7.5 years in human terms
>
```

Evidence 10

(untitled)

```
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? cat
enter the cat's age in years? 1.5
Your cat is 20 years in human terms
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? dog
enter the dog's age in years? 1.5
Your dog is 16.5 years in human terms
>
```

Evidence 11

(untitled)

```
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? cat
enter the cat's age in years? -2
? 1.7
Your cat is 22 years in human terms
>RUN
Cat and Dog age to human equivalent
Choose Cat or Dog, type cat or dog? dog
enter the dog's age in years? -3.98
? 6.78
Your dog is 41.12 years in human terms
>_
```

Evaluation:

Tested with numeric data, results as expected, seems to work just as well with decimal values so these have been included.

Fixed issue with non-numeric and zero or negative values with simple repeat loop to reject these values.

```
PRINT "Cat and Dog age to human equivalent"
  REPEAT
   INPUT "Choose Cat or Dog, type cat or dog", animal$
  UNTIL animal$ ="cat" OR animal$="dog"
  PRINT "enter the ";animal$;"'s age in years";

  REPEAT
   INPUTage
  UNTIL age >0

  IF animal$="dog" THEN
   IF age<=2 THEN
    animalage = age*11
   ELSE
    animalage = 22+(age-2)*4
   ENDIF
  ENDIF

  IF animal$="cat" THEN
   IF age <= 1 THEN
    animalage =age* 15
   ELSE
    IF age <= 2 THEN
     animalage =15 +(age-1)*10
    ELSE
     animalage= 25+(age-2)*4
    ENDIF
   ENDIF
  ENDIF

  PRINT "Your ";animal$;" is ";animalage;" years in human terms"
```

A453

Task 2: Password strength sample task

Analysis

1. Input a password
2. Is the password between 6 and 12 characters long?

    No; reject and return to stage 1

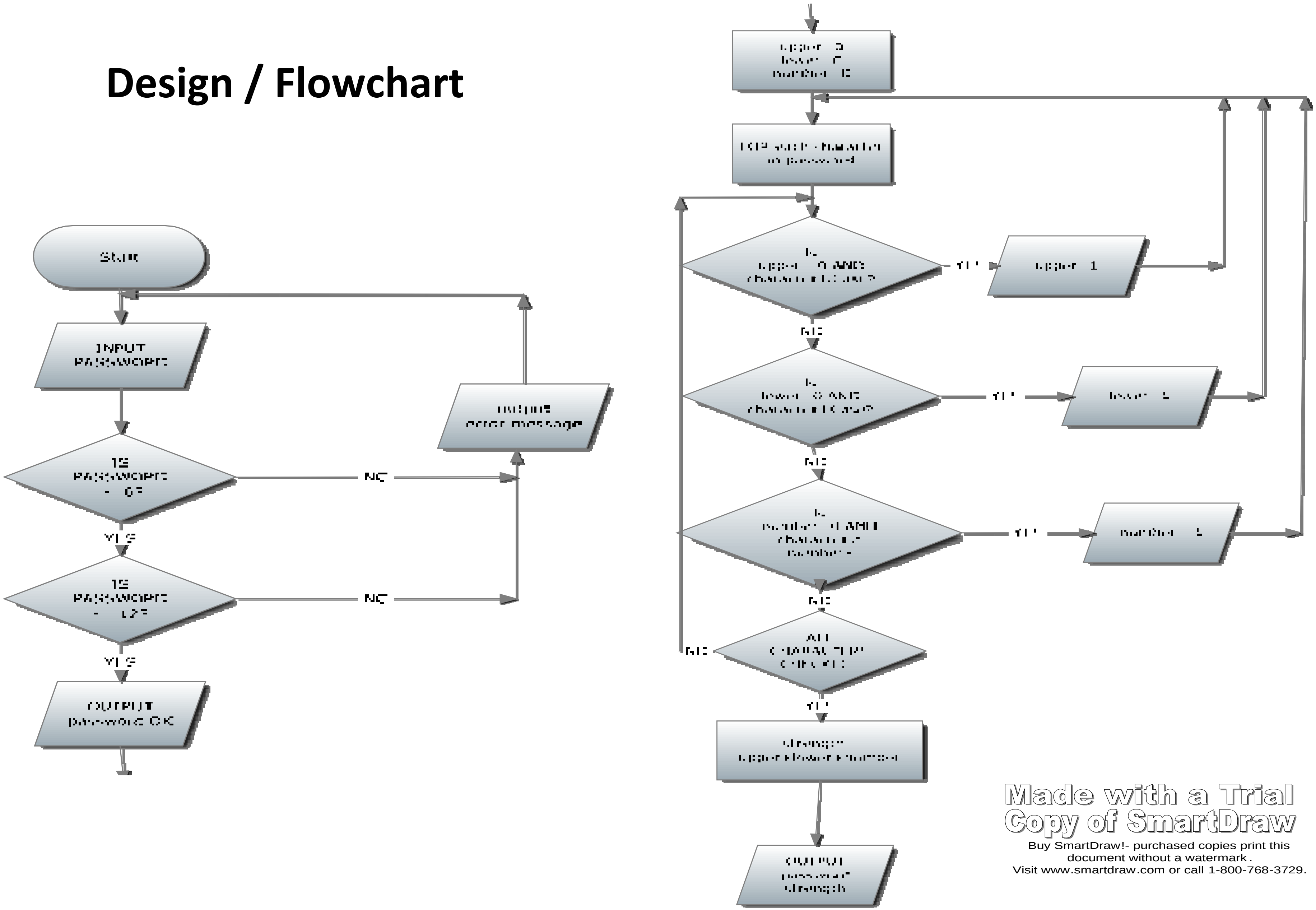
    YES; output message and carry on
3. Check each character of the password in turn

    Is this character upper case? If yes flag that upper case is included

    Is this character lower case? If yes flag that lower case is included

    Is this character a number? If yes flag that number is included
4. If three flags set then the password is STRONG
5. If two flags set then the passwords is MEDIUM
6. If one flag set then the password is WEAK

Design / Flowchart



Design / Pseudocode

```
REPEAT
INPUT the password
len=length of password
IF len <6 OR len >12 THEN
PRINT suitable error message
UNTIL len >=6 and <=12
PRINT password OK
Initialise upper, lower and number to 0
FOR i = 1 TO len
IF MID$(password, i, 1) is upper AND upper =0 THEN upper
=1
ELSE
IF MID$(password, i, 1) is lower AND lower =0 THEN lower
=1
ELSE
IF MID$(password, i, 1) is number AND number =0 THEN
number =1
NEXT i
strength = upper+lower+number
CASE
strength = 1 then PRINT “WEAK”
strength = 2 then PRINT “MEDIUM”
strength = 3 then PRINT “STRONG”
```

Design / Test strategy

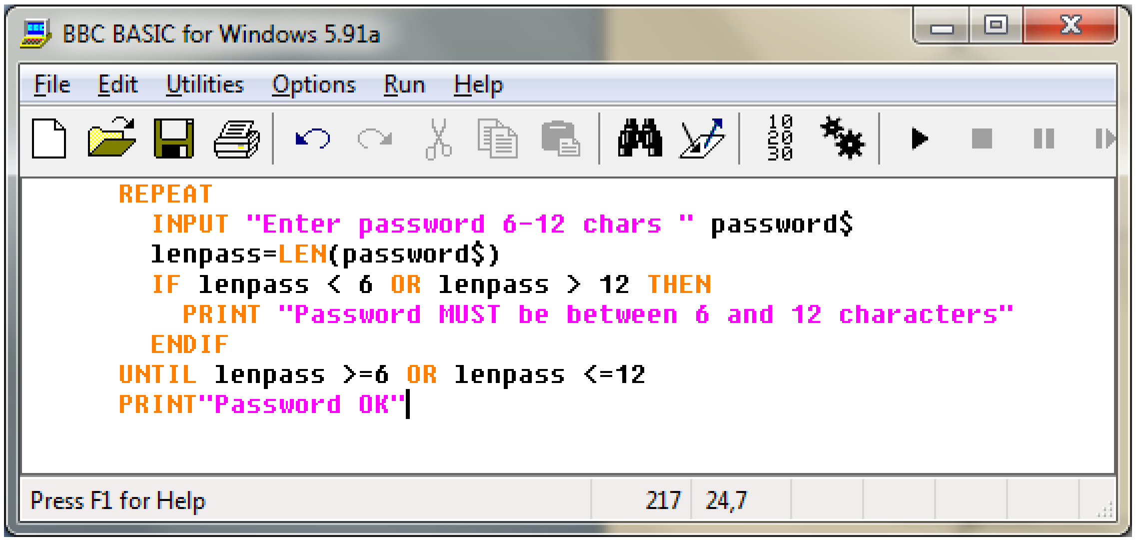
We need a test strategy to use during development to show that the solution works at each stage.
Requirement: 6- 12 characters

Less than 6	6 exactly	>6 and <12	12 exactly	More than 12
qwert	qwerty	qwertyui	qwertyuiopas	qwertyuiopasd
5 characters should be rejected	6 characters, boundary, should be OK	8 characters, valid input should be OK	12 characters, boundary, should be OK	13 characters should be rejected

Weak, Medium and Strong identified:

Weak	Medium			Strong
qwerty QWERTY 123456	Qwerty Awerty Zwerty	3werty 3WERTY Q23456	qwe456 1w3r5y QW345Y	QW34ty 12erTY
All lower / upper case/ numeric: weak reported	1 upper case, rest lower, also test A and Z accepted Medium reported	All medium strength combinations	With differing quantities of numbers and letters	Both cases and numeric used, Strong reported

Development & Testing



The code is being tested as it is developed, here the length is checked with 5, 6,12 and 13 char passwords.
The evidence shows that this section of the code works.

Design / Approach to testing for password strength

In my solution I am going to use the fact that all characters have a unique ASCII value
A is 65, B is 66 ... up to Y which is 90
a is 97, b is 98 ... up to y which is 122
0 is 48, 1 is 49 ... Up to 9 which is 57

So if the ASCII value of each character in the password is checked then we can identify if it is upper, lower or numeric.

Using LEN to check the length of the password and a simple loop from 1 to LEN(password) with the MID$ command I can check each character individually

If I identify an upper case I will set a flag once
Similarly with lower case and numeric.

If only 1 flag is set it will be WEAK, 2 it will be MEDIUM, 3 it will be STRONG

Development & Testing

BBC BASIC for Windows 5.91a

File Edit Utilities Options Run Help

```
REPEAT
  INPUT "Enter password 6-12 chars " password$
  lenpass=LEN(password$)
  IF lenpass < 6 OR lenpass > 12 THEN
    PRINT "Password MUST be between 6 and 12 characters"
  ENDIF
UNTIL lenpass >=6 OR lenpass <=12
PRINT"Password OK"
upper=0
lower=0
number=0
FOR i =1 TO lenpass
  IF ASC(MID$(password$,i,1))>=65 AND  ASC(MID$(password$,i,1))<=90 THEN
    IF upper=0 THEN
      upper = 1
    ENDIF
  ENDIF
NEXT i
PRINT upper
```

Press F1 for Help

password code

```
Enter password 6-12 chars qwerty
Password OK
         0
```

password code

```
Enter password 6-12 chars Awerty
Password OK
         1
>
```

password code

```
Enter password 6-12 chars Zwerty
Password OK
         1
```

Add the next section of code to check for upper case and test it with suitable test data: for example

qwerty, should return 0

Awerty should return 1 and checks that A is included in the range

Zwerty should return 1 and checks that Z is included etc.

Similarly for lower case and numeric data in the test strategy

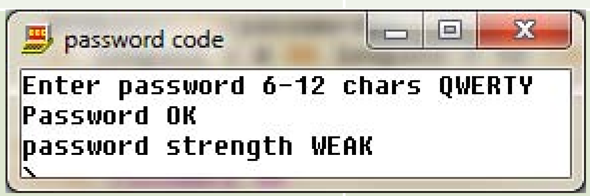
Development & Testing

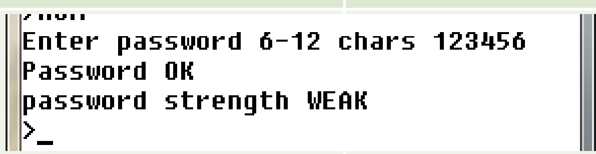
The code that was used and tested for upper case is simply copied and pasted then modified accordingly and checked at each stage.

```
REPEAT
  INPUT "Enter password 6-12 chars " password$
  lenpass=LEN(password$)
  IF lenpass < 6 OR lenpass > 12 THEN
    PRINT "Password MUST be between 6 and 12 characters"
  ENDIF
UNTIL lenpass >=6 OR lenpass <=12
PRINT"Password OK"
upper=0
lower=0
number=0
FOR i =1 TO lenpass
  IF ASC(MID$(password$,i,1))>=97 AND  ASC(MID$(password$,i,1))<=122 THEN
    IF lower=0 THEN
      lower = 1
    ENDIF
  ENDIF
NEXT i
FOR i =1 TO lenpass
  IF ASC(MID$(password$,i,1))>=65 AND  ASC(MID$(password$,i,1))<=90 THEN
    IF upper=0 THEN
      upper = 1
    ENDIF
  ENDIF
NEXT i
FOR i =1 TO lenpass
  IF ASC(MID$(password$,i,1))>=48 AND  ASC(MID$(password$,i,1))<=57 THEN
    IF number=0 THEN
      number = 1
    ENDIF
  ENDIF
NEXT i
strength=upper+lower+number
CASE strength OF
  WHEN 1 : PRINT "password strength WEAK"
  WHEN 2 : PRINT "password strength MEDIUM"
  WHEN 3 : PRINT "password strength STRONG"
ENDCASE
```

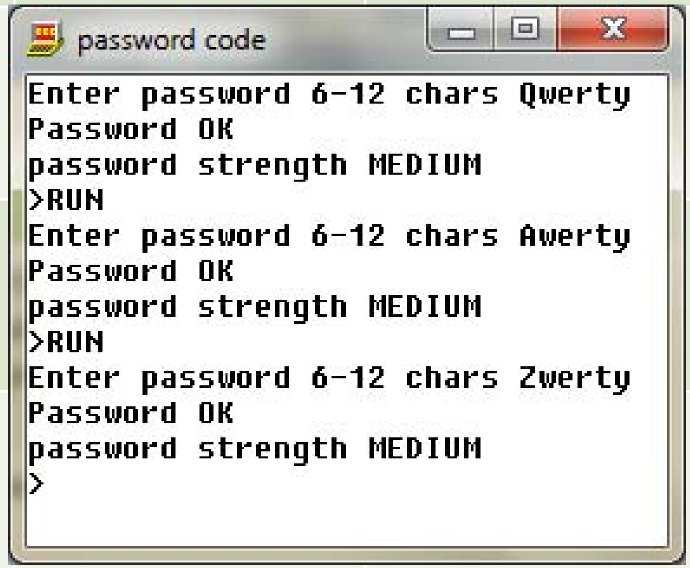
Testing

Weak	Medium			Strong
All lower / upper case/ numeric: weak reported	1 upper case, rest lower, also test A and Z accepted Medium reported	All medium strength combinations	With differing quantities of numbers and letters	Both cases and numeric used, Strong reported
qwerty ✓			qwe456	QW34ty
QWERTY ✓			1w3r5y	12erTY
123456 ✓			QW345Y	

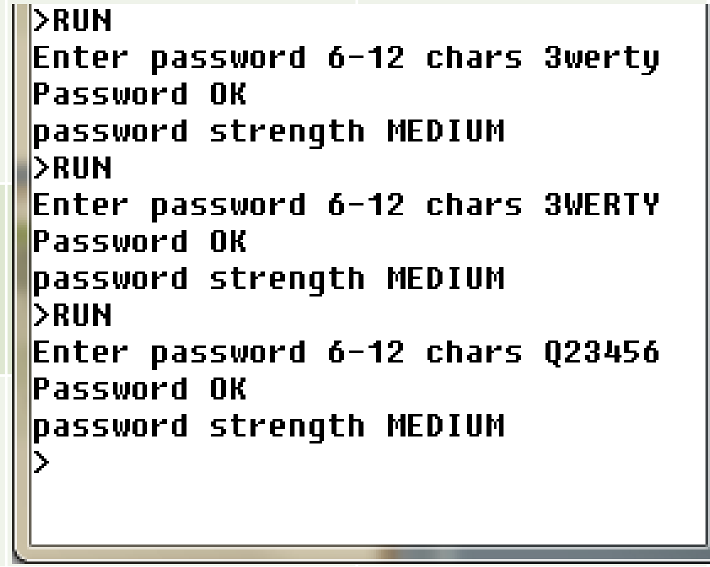




Weak	Medium			Strong
All lower / upper case/ numeric: weak reported	1 upper case, rest lower, also test A and Z accepted Medium reported	All medium strength combinations	With differing quantities of numbers and letters	Both cases and numeric used, Strong reported
qwerty ✓	Qwerty ✓			QW34ty
QWERTY ✓	Awerty ✓			12erTY
123456 ✓	Zwerty ✓			

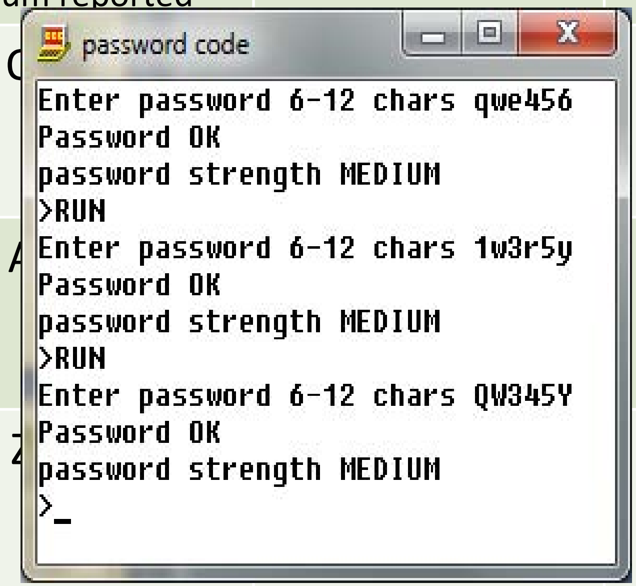


Weak	Medium			Strong
All lower / upper case/ numeric: weak reported	1 upper case, rest lower, also test A and Z accepted Medium reported	All medium strength combinations	With differing quantities of numbers and letters	Both cases and numeric used, Strong reported
qwerty ✓	Qwerty ✓	3werty ✓		
QWERTY ✓	Awerty ✓	3WERTY ✓		
123456 ✓	Zwerty ✓	Q23456 ✓		

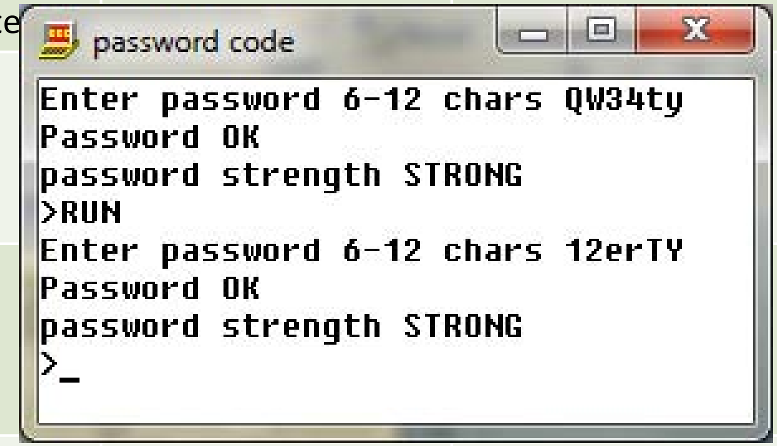


```
>RUN
Enter password 6-12 chars 3werty
Password OK
password strength MEDIUM
>RUN
Enter password 6-12 chars 3WERTY
Password OK
password strength MEDIUM
>RUN
Enter password 6-12 chars Q23456
Password OK
password strength MEDIUM
>
```

Weak	Medium			Strong
All lower / upper case/ numeric: weak reported	1 upper case, rest lower, also test A and Z accepted Medium reported	All medium strength combinations	With differing quantities of numbers and letters	Both cases and numeric used, Strong reported
qwerty ✓	C		qwe456 ✓	QW34ty
QWERTY ✓	A		1w3r5y ✓	12erTY
123456 ✓	Z		QW345Y ✓	



Weak	Medium			Strong
All lower / upper case/ numeric: weak reported	1 upper case, rest lower, also test A and Z accepted Medium reporte	All medium strength combinations	With differing quantities of numbers and letters	Both cases and numeric used, Strong reported
qwerty ✓	Qwerty ✓			QW34ty ✓
QWERTY ✓	Awerty ✓			12erTY ✓
123456 ✓	Zwerty ✓	Q23456 ✓	QW345Y ✓	



Testing complete all tests worked as expected.

Code explained

```
REPEAT
  INPUT "Enter password 6-12 chars " password$
  lenpass=LEN(password$)
  IF lenpass < 6 OR lenpass > 12 THEN
    PRINT "Password MUST be between 6 and 12 characters"
  ENDIF
UNTIL lenpass >=6 OR lenpass <=12
PRINT"Password OK"
upper=0
lower=0
number=0
FOR i =1 TO lenpass
  IF ASC(MID$(password$,i,1))>=97 AND  ASC(MID$(password$,i,1))<=122 THEN
    IF lower=0 THEN
      lower = 1
    ENDIF
  ENDIF
NEXT i
FOR i =1 TO lenpass
  IF ASC(MID$(password$,i,1))>=65 AND  ASC(MID$(password$,i,1))<=90 THEN
    IF upper=0 THEN
      upper = 1
    ENDIF
  ENDIF
NEXT i
FOR i =1 TO lenpass
  IF ASC(MID$(password$,i,1))>=48 AND  ASC(MID$(password$,i,1))<=57 THEN
    IF number=0 THEN
      number = 1
    ENDIF
  ENDIF
NEXT i
strength=upper+lower+number
CASE strength OF
  WHEN 1 : PRINT "password strength WEAK"
  WHEN 2 : PRINT "password strength MEDIUM"
  WHEN 3 : PRINT "password strength STRONG"
ENDCASE
```

Get password
Find length of password
Check length >=6, <=12
If not print error message

If error return to get password otherwise
Print OK

Initialise variables

By checking each character in the string for ASCII values
97-122 for a-z,
Identify if the character is lower,
If one of these has not already been found flag by setting the variable to 1, otherwise ignore.

Repeat process for upper and number using
65-90 for A-Z and
48-57 for 0-9

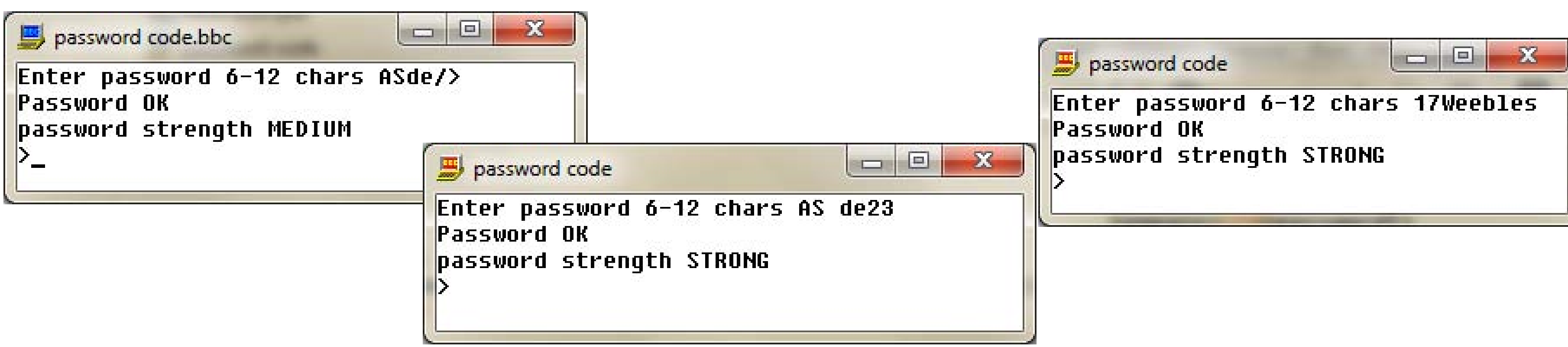
Add together the values for lower, upper and number to get a strength value.
Use the CASE command to respond according to the numeric value

Evaluation of the solution

We now have basic functionality but we need to complete some final product testing with a range of data and typical end users.

Further testing is completed with a range of valid and invalid data:

Test	Data	Expected	Actual / Comment
To see what happens with special characters	ASde/>	Since /> are not checked they will not set any flag hence MEDIUM	Medium, as expected: Code should be modified to reject special characters
To see what happens if spaces are used	AS de23	Since space has an ASCII value it will be accepted and rated as STRONG	Strong as expected: Code should not accept space and should be modified
Typical strong password	17Weebles	Strong, valid data	Strong.



Feedback from user testing

The whole point of a password is for security, the password is displayed when typed in, this is a problem.

The code should be modified to display * characters instead of the input values.

Evaluation of the solution

Using ASCII values to check the case etc works well and this could be extended to reject non alphanumeric characters by examining the password after input for characters out of range returning the user to the input screen with a suitable error message.

From the test data provided it is clear that the code segment meets the basic requirements:
6-12 characters
Upper, lower and numeric cases through the use of flags can identify weak, medium or strong passwords.

Also from the testing it can be seen that the system also accepts spaces and special characters since, though does not flag any value to these. The code should be modified accordingly to reject these.

The testing also suggests that the password input should be masked if it is to be of any real value, such a modification can easily be completed by overwriting the input area with * characters.
This code is functional and could be used as a module in a larger program if suitably modified as identified in the test section.

Evaluation of the solution

- Possible improvements:
  - Check data on entry for character types.
  - Check data on entry for invalid characters.
  - blanking the password on entry by replacing the characters with *'s
  - providing a more interesting or 'friendly' interface

A453

Task 3: High scores table sample task

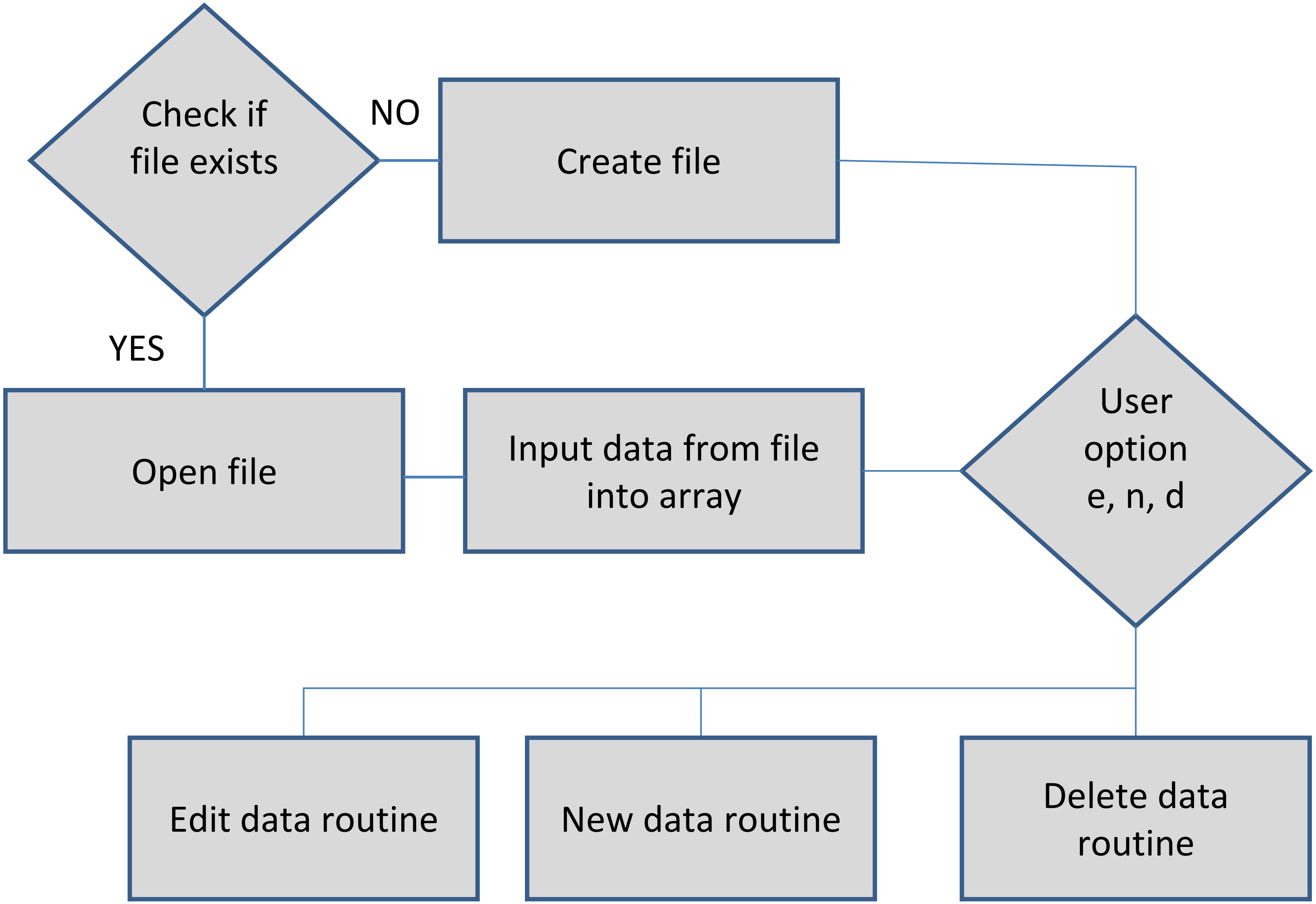
Analysis

- A system to manage high scores
  - Create a file and be able to
  - Find a score for a user
  - Update a score for a user
  - Add a new user and score
  - Delete a user and score

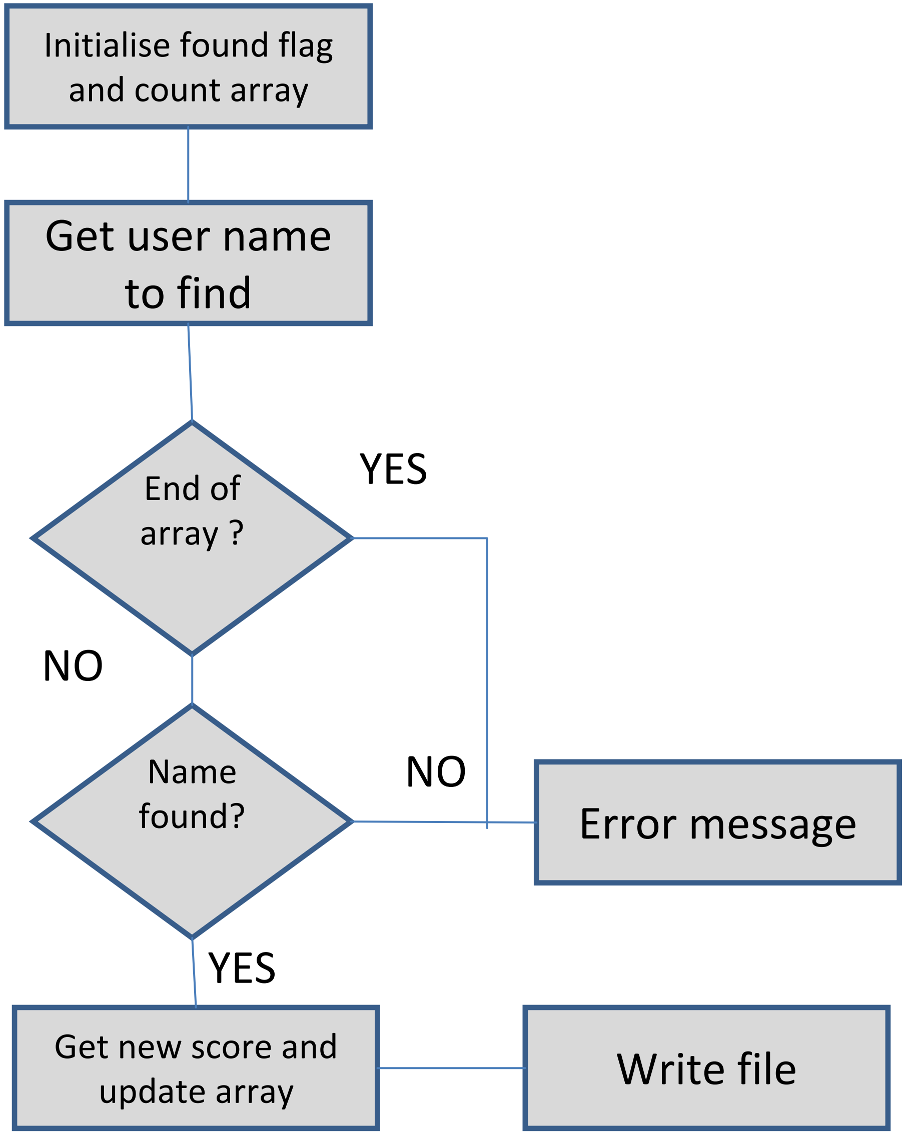
Analysis

- Need to check if the file exists, if not create one
- Need to load data from the file into an array
- Need to check if username exists to update score, if not error message
- Need an option system for edit, new and delete, otherwise error message
- Need to write modified data back to file

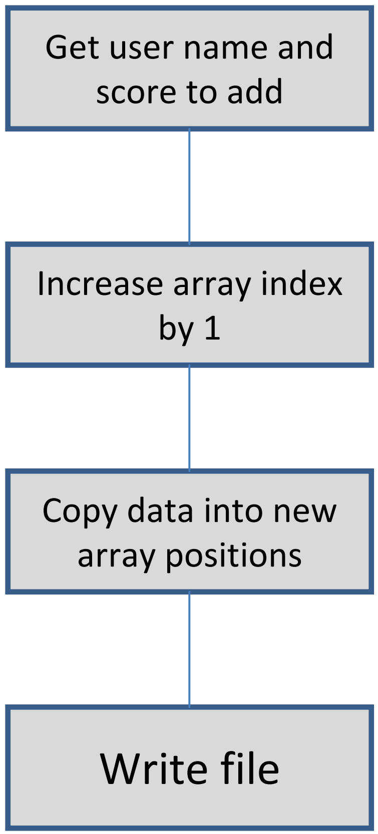
Design / File exists – choose option

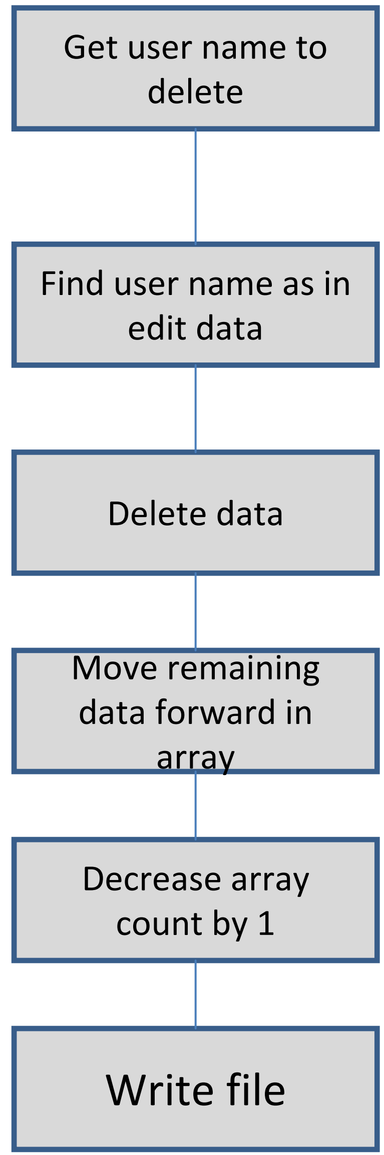


Design / Edit data option



Design / New data option and delete option



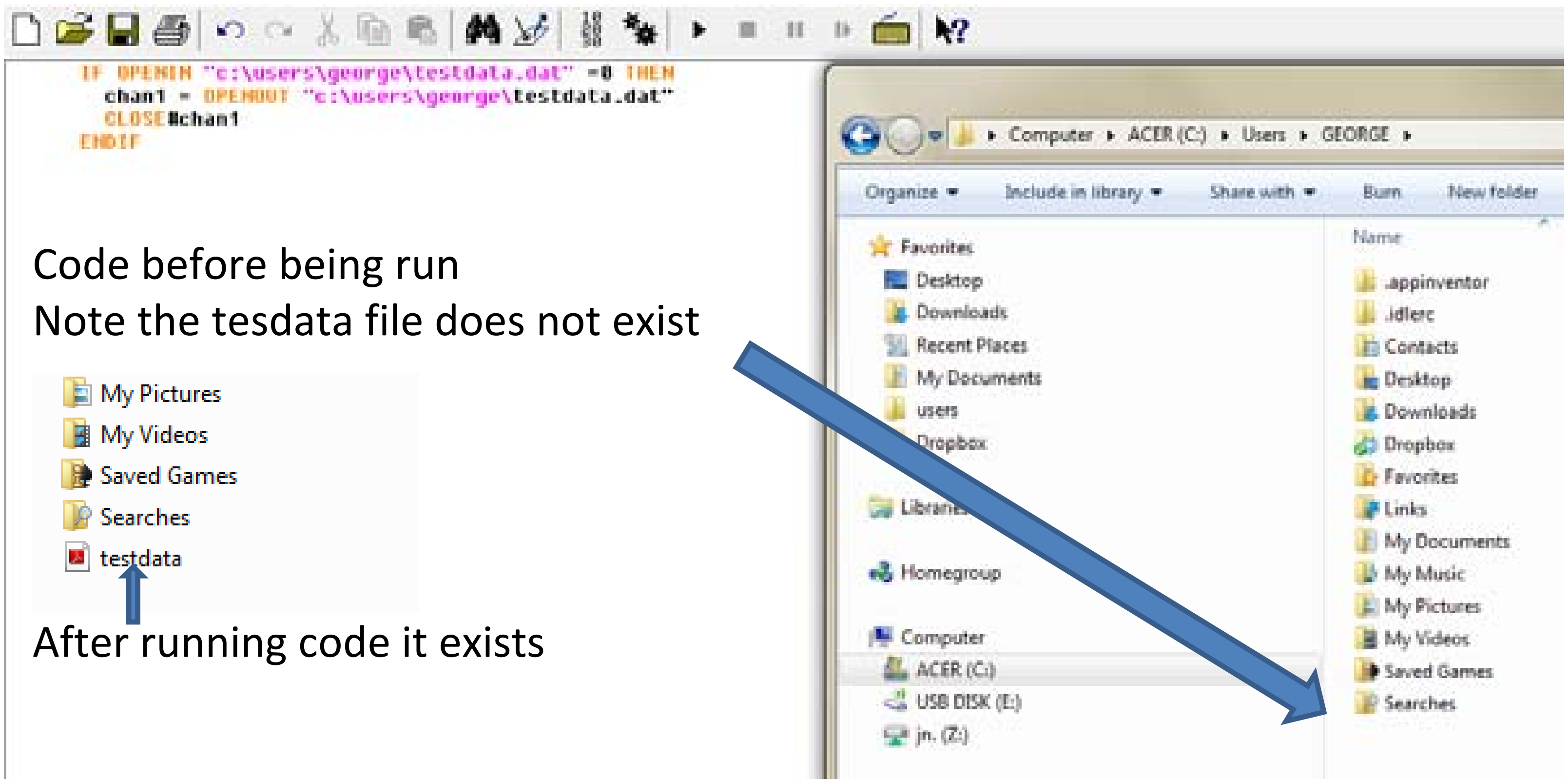


Design / development

Check file exists: if I try to open a file it returns 0 if the file doesn't exist so I can use this to decide if I need to create a file.

If openin file = 0 the create file else open file

```
IF OPENIN "c:\users\george\data.dat" =0 THEN
    chan1 = OPENOUT "c:\users\george\data.dat"
    CLOSE#chan1
  ENDIF
```

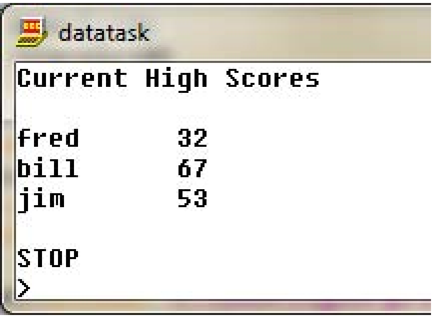


Design / development

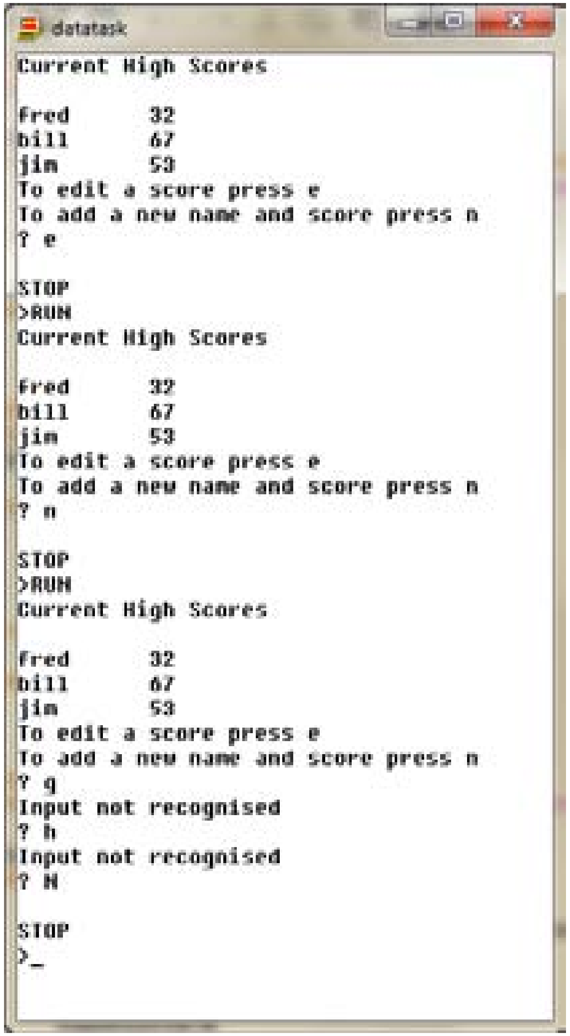
I need to create an array for the names and scores so that I can read in the data from a file.

I have created a simple text file with three names and scores to test this section of code.

```
DIM name$(10)
DIM score$(10)

    index=1
    IF OPENIN "c:\users\george\data.dat" =0 THEN
      chan1 = OPENOUT "c:\users\george\data.dat"
      CLOSE#chan1
    ENDIF

    chan1=OPENIN "c:\users\george\data.dat"
    REPEAT
      INPUT#chan1,name$(index)
      INPUT#chan1,score$(index)
      index=index+1
    UNTIL EOF#chan1
    CLOSE#chan1
    index=index-1

    PRINT "Current High Scores“
FOR i = 1 TO index
      PRINT name$(i),score$(i)
    NEXT i
```



Design / development



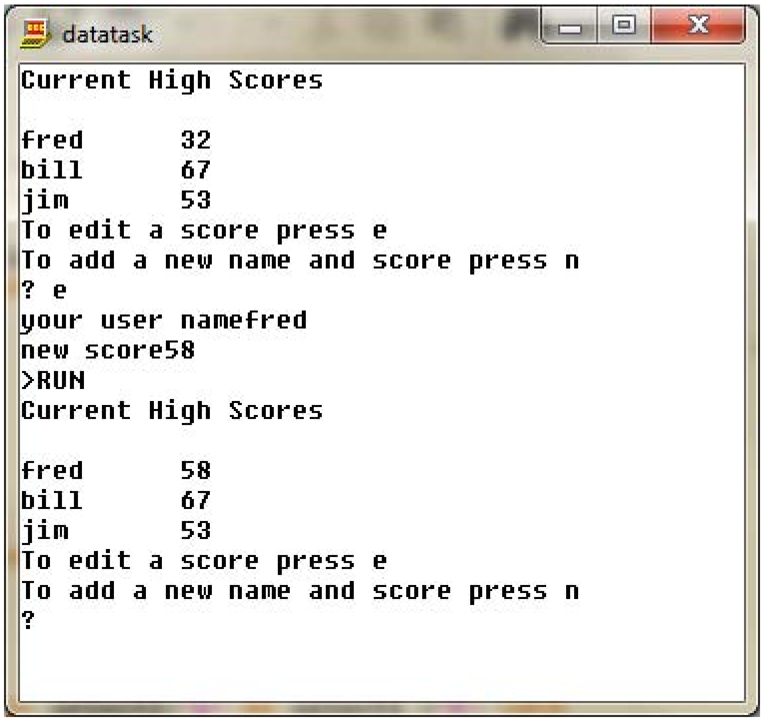
I need the user to be able to select one of the options. I will do edit and new first.

```
PRINT "To edit a score press e"
    PRINT "To add a new name and score press n"

    inputvalid=0
    REPEAT
      INPUT select$
      IF select$ = "e" OR select$ = "E" OR select$ = "n" OR select$ = "N" THEN
        inputvalid=1
      ELSE
        PRINT "Input not recognised"
        inputvalid=0

      ENDIF
    UNTIL inputvalid=1
```

The flag inputvalid is used to end the loop if a valid input is entered but repeat the process until a valid input is entered. I used the OR to allow for e, E, n OR N inputs.

I used e, n, g, h, N. e,n and N were accepted but g and h made the loop request input again

Design / development

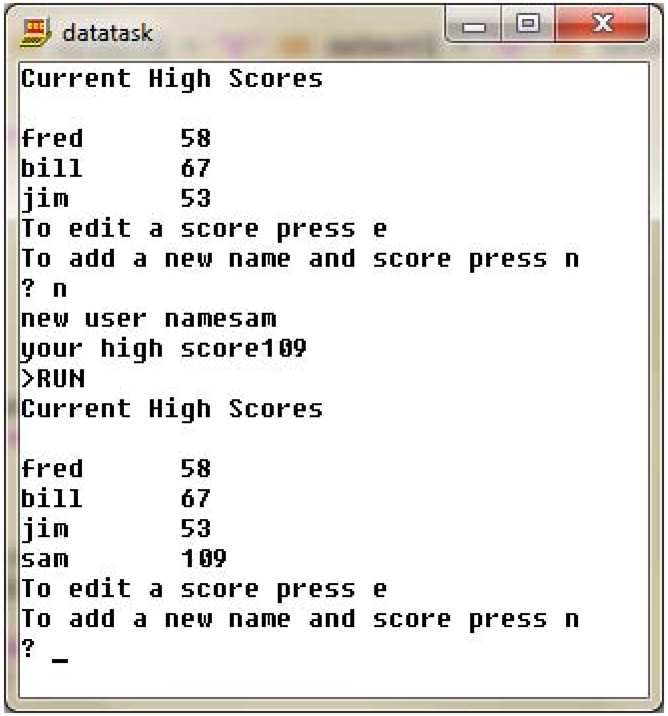


The data for fred has been updated as expected from 32 to 58.

The edit routine should search for the user name in the array, set a flag if found, edit the score and write the new score to the array. If not found it should print and error message.
It checks not found by checking the found flag and the count through the array compared to how many were read in.

```
IF select$="e" OR select$ ="E" THEN
    INPUT "your user name" user$

    flag=0
    count=1
    WHILE flag=0 AND count<=index
      IF user$= name$(count) THEN
        flag=1
        INPUT "new score" newscore$
        score$(count) = newscore$
      ENDIF
      IF flag=0 AND count>=index THEN
        PRINT "user name not found"
      ENDIF
      count=count+1
    ENDWHILE
ENDIF
```

Design / development

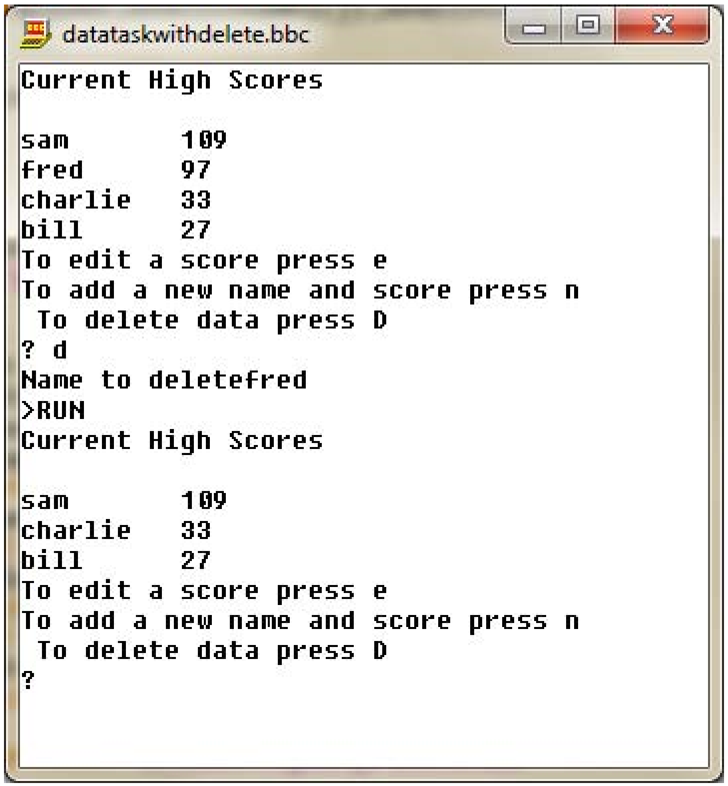


The data for sam has been added at the end of the array as expected.

To add the new data feature I will replace the endif with an else that allows new data to be added.

```
IF select$="e" OR select$ ="E" THEN
    INPUT "your user name" user$

    flag=0
    count=1
    WHILE flag=0 AND count<=index
     IF user$= name$(count) THEN
      flag=1
      INPUT "new score" newscore$
      score$(count) = newscore$
     ENDIF
     IF flag=0 AND count>=index THEN
      PRINT "user name not found"
     ENDIF
     count=count+1
    ENDWHILE
ELSE
    INPUT "new user name" newname$
    INPUT "your high score" highscore$
    index=index+1
    name$(index)= newname$
    score$(index)= highscore$
   ENDIF
```

Design / development



The data for fred has been deleted from the file as expected.

To delete an item is more complicated, but if I just rewrite the list back to file skipping the deleted name then will effectively delete the user

```
IF select$="d" OR select$ ="D" THEN
    INPUT "Name to delete" delete$
    newcount=0
    FOR i=1 TO index
     IF delete$=name$(i) THEN
     NEXT i
    ELSE
     newcount=newcount+1
     newlistname$(newcount)=name$(i)
     newlistscore$(newcount)=score$(i)
    NEXT i
  ENDIF

  chan2=OPENOUT "c:\users\george\data.dat"
  FOR j=1 TO newcount
    PRINT#chan2,newlistname$(j),newlistscore$(j)
  NEXT j
  CLOSE#chan2

  ENDIF
```

I needed two new arrays and a new counting variable to do this.

Design / development

To make the final changes to include the delete option I created three IF THEN sections with each writing the modified data to the file.
I added two new arrays
This section shows initialising arrays checking if file exists
And printing high score table

```
REM Initiaalise arrays
    DIM name$(10)
    DIM score$(10)
    DIM newlistname$(10)
    DIM newlistscore$(10)

    REM check if file exists, if not create one
    index=1
    IF OPENIN "c:\users\george\data.dat" =0 THEN
      chan1 = OPENOUT "c:\users\george\data.dat"
      CLOSE#chan1
    ENDIF

    REM read in data from file and display
    chan1=OPENIN "c:\users\george\data.dat"
    REPEAT
      INPUT#chan1,name$(index)
      INPUT#chan1,score$(index)
      index=index+1
    UNTIL EOF#chan1
    CLOSE#chan1
    index=index-1

    PRINT "Current High Scores"
    FOR i = 1 TO index
      PRINT name$(i),score$(i)
    NEXT i
```

Design / development

This section shows getting user input and validating user input

```
 REM get user input and validate
     PRINT "To edit a score press e"
     PRINT "To add a new name and score press n"
     PRINT " To delete data press D"

     inputvalid=0
     REPEAT
       INPUT select$
       IF select$ = "e" OR select$ = "E" OR select$ = "n" OR select$ = "N" OR
select$= "D" OR select$="d" THEN
         inputvalid=1
       ELSE
         PRINT "Input not recognised"
         inputvalid=0

       ENDIF
     UNTIL inputvalid=1
```

Design / development

This section is the delete option
•It get the name to delete
•Compares the data in the array to this value
•If the values match it skips to the next item
•If they don't match it writes the old data into a new array and
•counts entries into this new array

```
REM delete option write to new array skipping name and score to delete
  IF select$="d" OR select$ ="D" THEN
   INPUT "Name to delete" delete$
   newcount=0
   FOR i=1 TO index
    IF delete$=name$(i) THEN
    NEXT i
   ELSE
    newcount=newcount+1
    newlistname$(newcount)=name$(i)
    newlistscore$(newcount)=score$(i)
   NEXT i
  ENDIF
  REM write modified file back to disk
  chan2=OPENOUT "c:\users\george\data.dat"
  FOR j=1 TO newcount
   PRINT#chan2,newlistname$(j),newlistscore$(j)
  NEXT j
  CLOSE#chan2

  ENDIF
```

Design / development

This is the edit section
It compares name to edit with data in the array and allows the user to retype the values for that entry.

```
REM edit option
  IF select$="e" OR select$ ="E" THEN
  INPUT "your user name" user$

  flag=0
  count=1
  WHILE flag=0 AND count<=index
   IF user$= name$(count) THEN
    flag=1
    INPUT "new score" newscore$
    score$(count) = newscore$
   ENDIF
   IF flag=0 AND count>=index THEN
    PRINT "user name not found"
   ENDIF
   count=count+1
  ENDWHILE

  chan2=OPENOUT "c:\users\george\data.dat"
  FOR count=1 TO index
   PRINT#chan2,name$(count),score$(count)
  NEXT count
  CLOSE#chan2
  ENDIF
```

Design / development

This is the new data section
It asks for new name and score then appends these to the end of the array and writes the data to file.

```
REM new name and score option
  IF select$="n" OR select$="N" THEN

  INPUT "new user name" newname$
  INPUT "your high score" highscore$
  index=index+1
  name$(index)= newname$
  score$(index)= highscore$

  chan2=OPENOUT "c:\users\george\data.dat"
  FOR count=1 TO index
    PRINT#chan2,name$(count),score$(count)
  NEXT count
  CLOSE#chan2
  ENDIF
```

Testing and evaluation

The testing was completed as the system was developed, see evidence of each section being tested during development.
The system does what was required:
Create a file: Checks to see if file exists then creates or opens the file
Add data to file: Data added by writing new item to array and writing data back to file
Locate data by name and high score: Incomplete, it can locate data by name to modify or delete, but not by high score, sort routine not implemented for this.
Delete an item and score, completed data to delete simply skipped when data written to new array and new array data written back to file.
Locate and update a a high score: Can update a score by user name

Most elements completed successfully but the interface is not clear and there are very limited user instructions or validation apart from choice of options and existence of data file.
Scores stored as string variables for convenience so no arithmetic possible but so sorting for highest score would require data to be converted to numeric values.