RECOGNISING ACHIEVEMENT

Candidate Style Answers

OCR GCSE ICT J461 / J061

Unit B065 Practical Applications in ICT: Controlled Assessment Task

Mid-Level Candidate Response

This support material booklet is designed to accompany the OCR GCSE ICT specifications for teaching from September 2010.

Introduction

OCR has produced these candidate style answers to support teachers in interpreting the assessment criteria for the new GCSE specifications and to bridge the gap between new specification release and availability of exemplar candidate work.

This content has been produced by subject experts, with the input of Chairs of Examiners, to illustrate how the sample assessment questions might be answered and provide some commentary on what factors contribute to an overall grading. The candidate style answers are not written in a way that is intended to replicate student work but to demonstrate what a "good" or "excellent" response might include, supported by examiner commentary and conclusions.

As these responses have not been through full moderation and do not replicate student work, they have not been graded and are instead, banded "middle" or "high" to give an indication of the level of each response.

Please note that this resource is provided for advice and guidance only and does not in any way constitute an indication of grade boundaries or endorsed answers.

Task

The task:

A teacher at a local primary school wants to help some students improve their spelling. These students will take time away from the class and work by themselves with a computer program that will help them to spell simple words from the class spelling lists for that week. The teacher wants to be able to specify the words for each student by entering them or by selecting them individually from a list.

The interface must be colourful and easy to use for the student; it needs to report back to the student on their success. The teacher also needs to know how well the student has done after each test by identifying words that caused problems and a score. The teacher interface needs to be simple to use.

Your task is to create a suitable solution with the basic functionality as a priority.

Analysis:

What do I know?

- My end user is a teacher in a primary school
- The system is to help 'some' students with their spelling
- The spellings are for simple words
- There is a class list for the week
- The teacher wants to select the words
- The interface should be colourful and easy to use
- The system should tell the student if they are right or not
- The teacher wants to see how well the student has done
- The teacher wants to know which words the student found difficult
- The system needs a simple teacher interface
- Basic functionality is the priority

What don't I know?

- The age of the students
- The ability of the students
- The type of word in the list
- The style of interface to be used
- How the system reports back to the student
- How the system reports back to the teacher
- How the teacher wants to input the words
- How the student can be identified by the system

Questions to ask the end user:

1. What age are the students in this group?
2. The system is for individual use, what is the reason for this?
3. What words are in the lists?
4. How do you want the system to ask for the spelling?
5. How do you want to input or select the words to be used?
6. How will students 'log in' to the system to keep scores or is that not necessary?

Questions to ask the end user:

1. What age are the students in this group?
2. The system is for individual use, what is the reason for this?
3. What words are in the lists?
4. How do you want the system to ask for the spelling?
5. How do you want to input or select the words to be used?
6. How will students 'log in' to the system to keep scores or is that not necessary?

Responses:

Q1. Aimed at young students just learning to spell, 4 to 6 years of age
- the basic vowel sound words

Q2. Individuals who need a little extra time, more practice while others get on with different work.

Q3,4. Basic vowel sound words such as short 'e', bed etc

-Q. The prompt for the words - spoken or picture or both?

-R The picture is what they will be used to but I like the idea of a spoken prompt as well - can we look at this while you develop the system?

-Q Could this activity be matching pictures to words?

-R possibly

-Q What about other interfaces such as hangman?

-R Not sure about hangman but there are other possibilities I have seen several - can you review these ideas and get back to me?

Q5. The words can be in a list for me to select some for a student or I can type them in, which ever is easiest.

-Q what about the spoken prompts, will you record them or shall we use a computer voice?

-R I like the idea of a computer voice if that is possible.

Q6 No need for a log in - I will set them up - all I need it for the system, to keep track of which ones they got right - I can do the rest.

Analysis of responses:

The system is for individual use by younger students. The teacher plans to set up the student in front of the computer with a list of words and wants the system to ask for spellings linked to pictures for alphabet words. The system should simply keep track of the words the student get correct so the teacher can deal with those they need extra practice with.

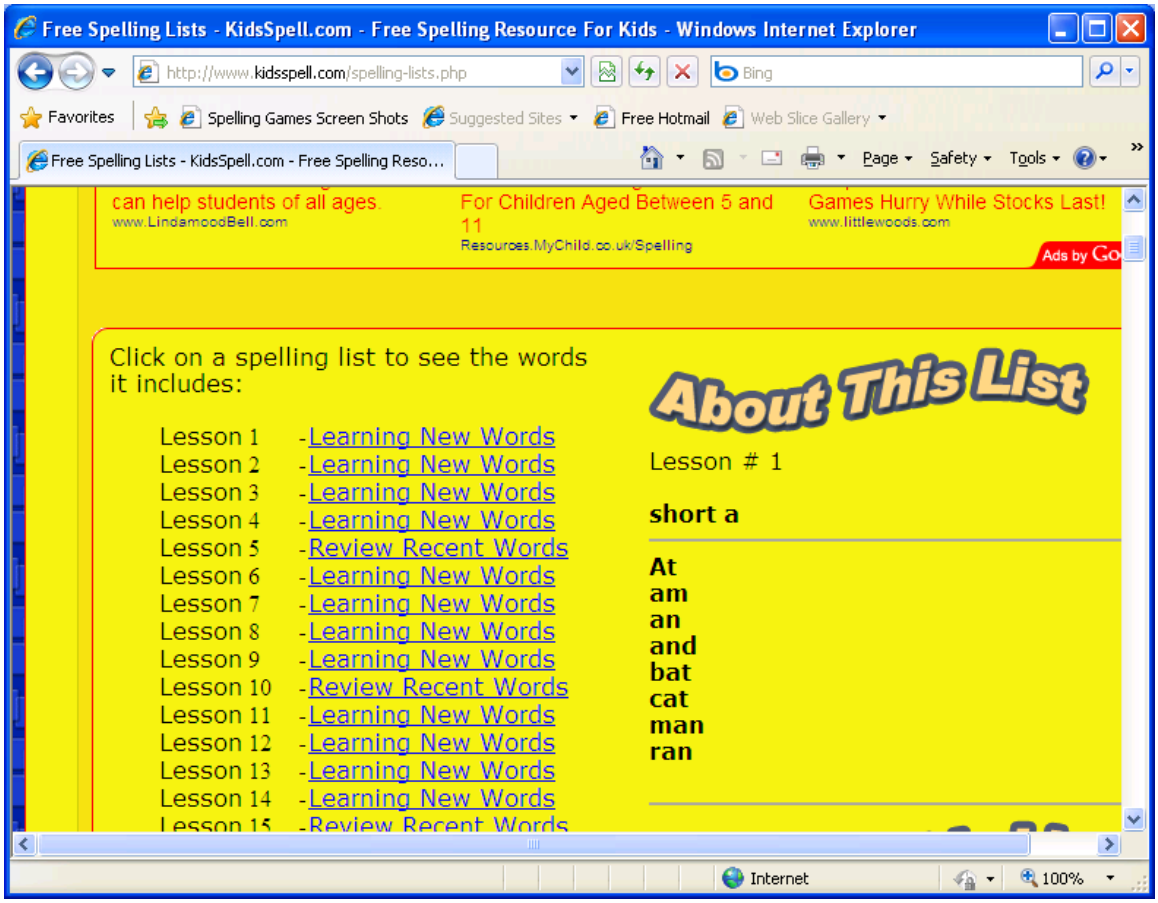
What I need to do now:

- I need to get a list of suitable words and locate suitable images.
- I need to decide on the interface style for the student by doing some research and getting back to the teacher with ideas to discuss.
- I need to think about how I can use spoken prompts.

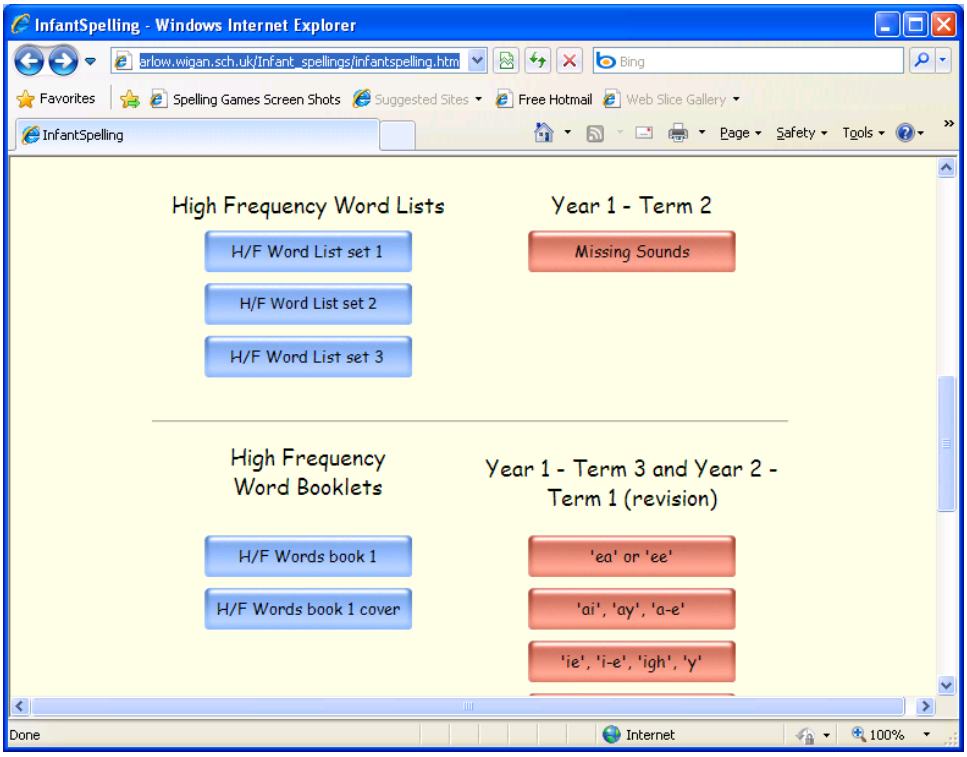
I shall search the internet for suitable word lists and suitable interface ideas.

Word lists:
Sites found include:

http://www.kidsspell.com/spelling-lists.php



http://www.saintambrosebarlow.wigan.sch.uk/Infant_spellings/infantspelling.htm


InfantSpelling - Windows Internet Explorer
arlow.wigan.sch.uk/Infant_spellings/infantspelling.htm
Favorites
Spelling Games Screen Shots
Suggested Sites
Free Hotmail
Web Slice Gallery
InfantSpelling
Page
Safety
Tools
High Frequency Word Lists
Year 1 - Term 2
H/F Word List set 1
H/F Word List set 2
H/F Word List set 3
Missing Sounds
High Frequency Word Booklets
Year 1 - Term 3 and Year 2 - Term 1 (revision)
H/F Words book 1
H/F Words book 1 cover
'ea' or 'ee'
'ai', 'ay', 'a-e'
'ie', 'i-e', 'igh', 'y'
Done
Internet
100%


ea, ee, y2 - Windows Internet Explorer
Favorites
Spelling Games Screen Shots
Suggested Sites
Free Hotmail
Web Slice Gallery
ea, ee, y2
Page
Safety
Tools
These words are spelt 'ee' or 'ea', but they all say 'ee'.
1.
feet
feat
2.
3.
Done
Internet
100%

Other sites I looked at included:
http://www.kidwaresoftware.com
http://www.bbc.co.uk/schools/ks1bitesize/literacy/spelling/index.shtml

These sites have interfaces that are complicated to develop and not required for my task but they do provide some ideas.

- Simple menus rather than complex GUI's appear to be more suited to my end users.
- Minimal choices, if any, are needed for the student user and a simple menu choice for the teacher.
- Words need to be based on sounds and the student needs to learn how to match the spellings to the sounds.
- Images are not very important since we are matching sounds and letters not learning the names for objects.

This means I need to investigate how to code spoken prompts for the words.

I have some experience with BBC Basic for Windows so I looked at various sites for ideas. www.bbcbasic.co.uk and www.bb4w.com which led to these links.

I could dismiss many of them immediately:

Everything EPOS

SimplEPOS *- a commercial point-of-sale and stock control package used in a broad range of retail businesses, written in BBC BASIC for Windows. Includes a downloadable demonstration version.*

Chris Hall

A software simulation of a signalbox with a description of how it was converted to run under Windows using BBC BASIC.

INSTAT

Instat *is a simple general statistics package. It started on the BBC Microcomputer in 1983, graduated to the PC in 1986 and then moved to Windows; it now uses BBC BASIC for Windows. Instat is intended to support the teaching of statistics, particularly for those who are not specialists.*

Simon Mathiassen

Striving to create some good games using *BBC BASIC for Windows* (including **Blast!** and **Unnamed Shooter**), but also has a few utilities.

Paw Books

For sale on this site is a **CD Book** containing over 100 *BBC BASIC for Windows* programs with answers and explanations. Also available are resources for digital logic design and simulation.

Planarsoft

Here you can download **BB4WMAPM**, a *BBC BASIC for Windows* library to perform high precision maths (up to 65000 digits) and **Calculator**, a high-precision scientific calculator using the BB4WMAPM library.

Jon Ripley

An introduction to writing *BBC BASIC* applications which make use of the Windows Application Program Interface, including a reference guide to data types, links to other useful documents, example code and libraries.

Dave Sergeant

Here you can download **Television Index for Windows**, a program written in *BBC BASIC for Windows* which provides a searchable index of past issues of **Television and Home Electronics Repair** magazine. Also assorted utilities for amateur radio and other applications.

Brian Stewart

Some small utilities for text manipulation and HTML generation. (Brian sadly died in 2007).

Gordon Sweet

A fairly large collection of old Games, Animations, BBC Music and other useful programs.

Tony Tooth

Various Programs in *BBC BASIC for Windows*, most of which use graphics to illustrate a mathematical problem (but you don't need to be a mathematician to run them!).

Richard Weston

On this site you will find a *BBC BASIC for Windows* beginners' tutorial, plus several example programs for free download, both as executable files and as BASIC program listings.

David Williams

This site contains some excellent arcade-style games written in *BBC BASIC for Windows*, including **Flak II**, **Bubballs**, **Buggy**, **Bugs**, **Crystal Hunter**, **Money Maze**, **Phroggy**, **Satsuma Panic**, **Spacerocks** and **SubZap**!

I looked at several of these sites and found a speak text routine on R T Russell's site and further examples of this being used in Gordon Sweet's site.

```
INSTALL @lib$+"COMLIBA"

ON ERROR PRINT 'REPORT$ : PROC_comexit : END
ON CLOSE PROC_comexit : QUIT

filetospeak$ = "C:\test.txt"

PROC_cominit
voice% = FN_createobject("SAPI.SpVoice")
PROC_callmethod(voice%, "Speak("""+filetospeak$+""", 12)")
PROC_releaseobject(voice%)
PROC_comexit
```

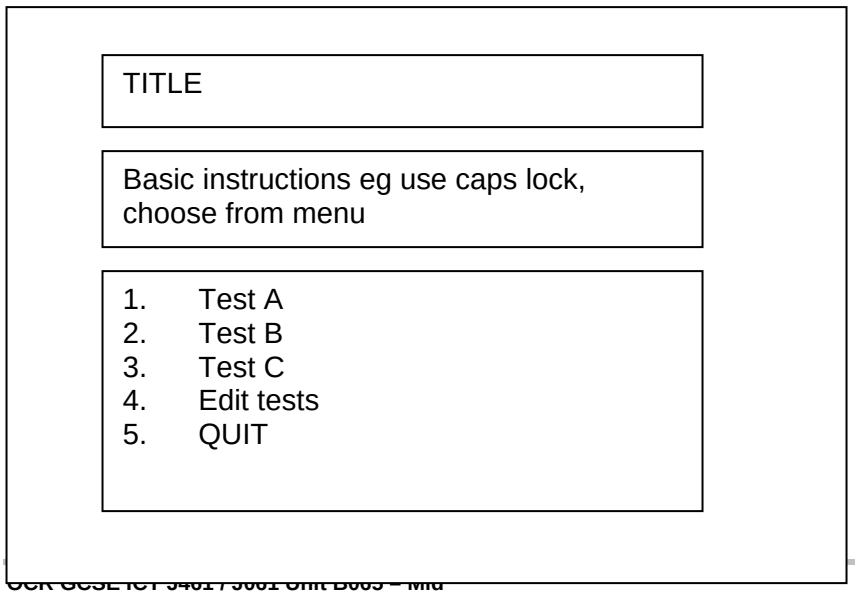
This will be my starting point for developing my program.

(There are various standard procedures to include as well as this code)

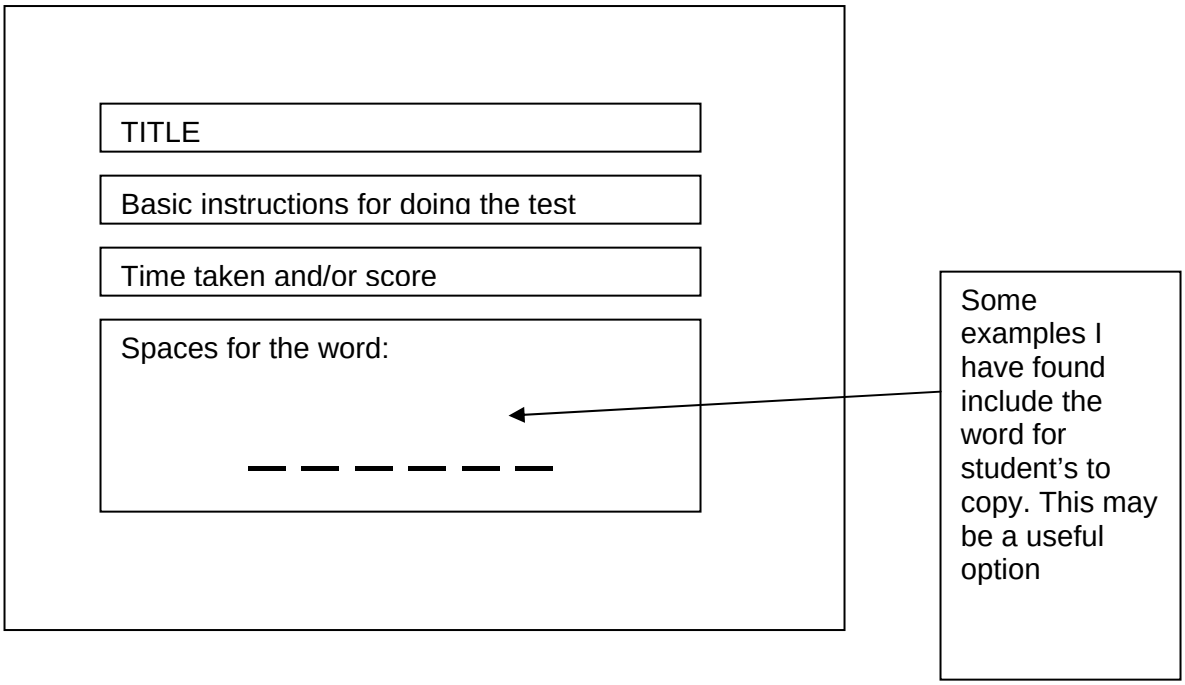
The teacher asked for a simple and colourful interface.

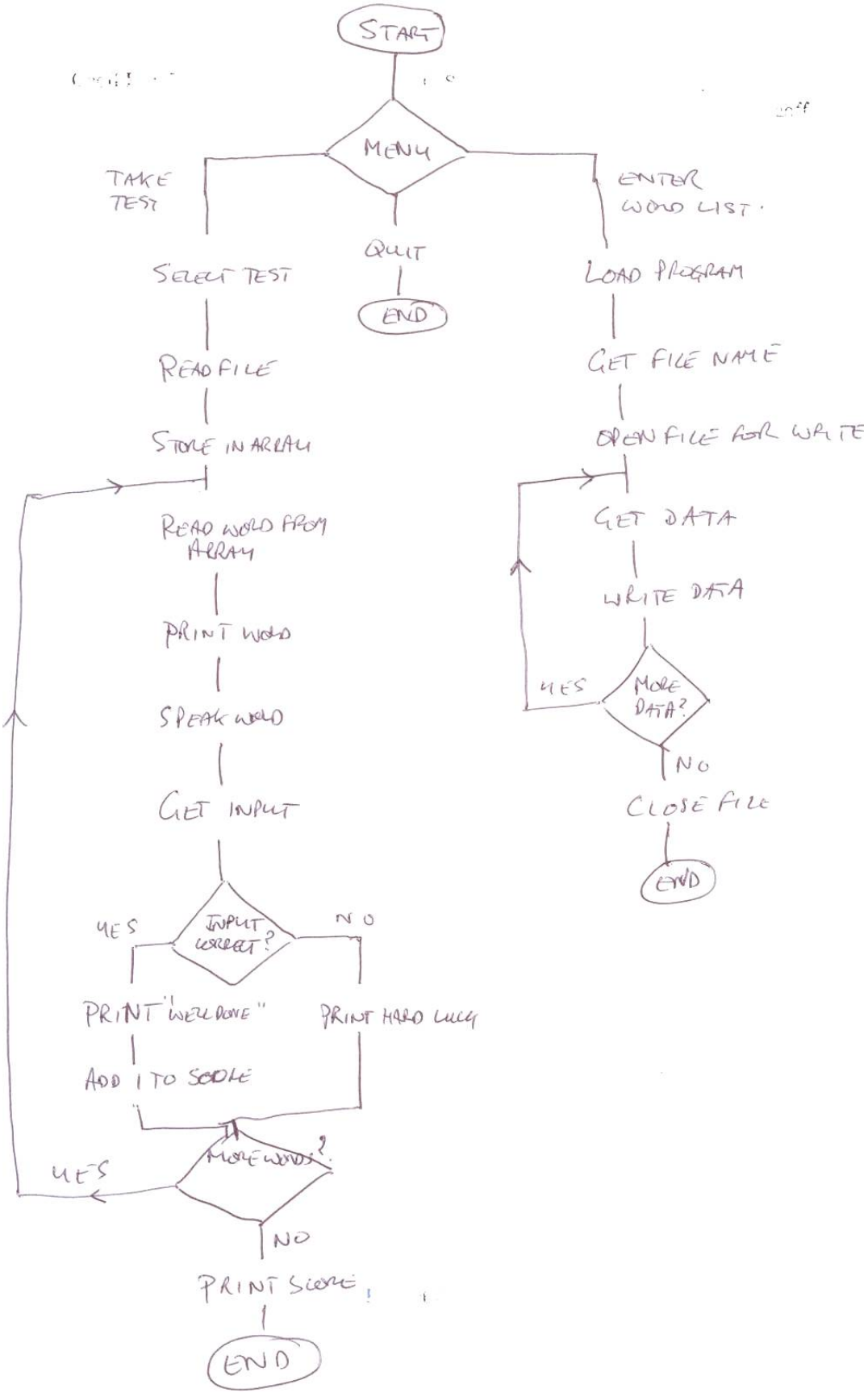
My research suggests a simple menu will be better than a GUI.

My suggestion is something like this for the menu



For the test interface




START
MENU
TAKE TEST
QUIT
ENTER WORD LIST
SELECT TEST
END
LOAD PROGRAM
READ FILE
GET FILE NAME
STORE IN ARRAY
OPEN FILE FOR WRITE
READ WORD FROM ARRAY
GET DATA
WRITE DATA
PRINT WORD
YES
MORE DATA?
SPEAK WORD
NO
GET INPUT
CLOSE FILE
END
YES
INPUT CORRECT?
NO
PRINT "WELL DONE"
PRINT HARD LUCK
ADD 1 TO SCORE
MORE WORDS?
YES
NO
PRINT SCORE
END

My proposal:

- A programmed solution using BBC Basic.
- A simple menu interface with 5 choices:
  - Three tests
  - Teacher option to modify the words
  - Quit
- The test interface keeps a score
- The system speaks the word using a computer voice
- The word may be displayed for the student to copy or not
- Simple space to type in the word
- Differing beeps for success or failure

Algorithms: (see above)

Speech procedure:

I have made some minor modifications to the speech procedure I found and have tested it with some normal data.

```
REM. based on program by R.T.Russell, 24-Apr-2007
REM. Error handling routine R.T Russell

INSTALL @lib$+"COMLIBA"

ON ERROR PROC_comexit : PRINT 'REPORT$ : END
ON CLOSE PROC_comexit : QUIT

PROC_cominit

Pitch% = 0
Speed% = 0
Voice$ = ""

REPEAT
  READ word$
  PRINT word$
  PROCspeak(word$,Pitch%,Speed%,Voice$)
UNTIL word$ = ""
PROC_comexit
END

DATA "Cat, Ben, Ball, Feet"
DATA "gtres, bonjour, *&£""
DATA

REM. Procedures from R T Russell

DEF PROCspeak(word$,pitch%,speed%,voice$)
tts% = FN_createobject("Sapi.SpVoice")
IF tts% THEN
  LOCAL qual$
  qual$ = "<PITCH ABSMIDDLE="""+STR$pitch%+"""/><RATE ABSSPEED="""+STR$speed%+"""/>"
  IF voice$<>"" qual$ += "<VOICE REQUIRED="""NAME="+voice$+"""/>"
  PROC_callmethod(tts%, "Speak("""+qual$+word$+""")")
  PROC_releaseobject(tts%)
  ENDPROC
ENDIF
tts% = FN_createobject("Speech.VoiceText")
IF tts% THEN
  PROC_callmethod(tts%, "Register("""","COMLIB demo")")
  PROC_putvalue(tts%, "Enabled(BTRUE)")
```

```
    PROC_putvalue(tts%, "Speed("+STR$INT(150*3^(speed%/10))+")")
    PROC_callmethod(tts%, "Speak("""+phrase$+""", 1)")
    REPEAT
      SYS "Sleep", 150
    UNTIL FN_getvalueint(tts%, "IsSpeaking") = 0
    PROC_releaseobject(tts%)
    ENDPROC
  ENDIF
```

Abnormal data for this unit would be words which are not standard English words or characters that are not alphabetic.

I used normal data Cat, Ben, Ball, Feet

Abnormal data included gtres, bonjour and *&£”

These abnormal data would probably be the result of typing errors but trapping them would be extremely difficult except for non alphabetic characters, to trap non-words would require the use of a dictionary function.

The normal data caused no problem, but the extra “ in the final abnormal data caused a problem since it was taken as an end “ on the data line leaving the final “ as an unfinished pair of “’s.

Error screen:

Rem0oving that item from the data and using just *&£ the system reads gtres as a word, bonjour works well and the *&£ are read as a string of symbols, ‘asterisk, ‘and’ ‘pound’

I will now modify the routine to use a file instead of the data lines.

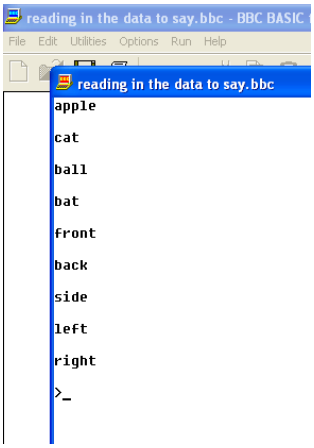
```
INSTALL @lib$+"COMLIBA"

ON ERROR PRINT 'REPORT$ : PROC_comexit : END
ON CLOSE PROC_comexit : QUIT

filetospeak$ = "C:\test.txt"

PROC_cominit
voice% = FN_createobject("SAPI.SpVoice")
PROC_callmethod(voice%, "Speak("""+filetospeak$+""", 12)")
PROC_releaseobject(voice%)
PROC_comexit
```

Using standard test data this is able to read a standard text file created using notepad.

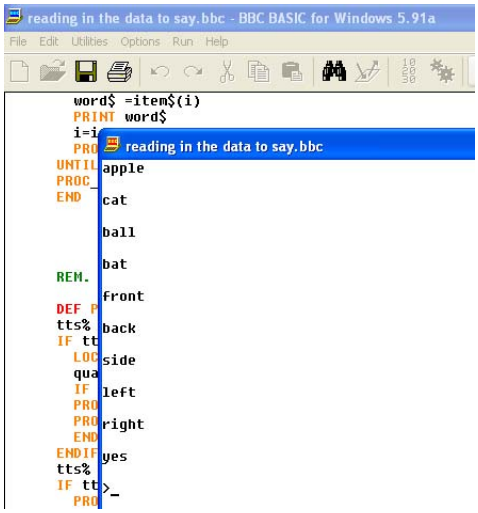
Now I need to get the system to speak individual words from the text file by detecting the breaks between the words. I have taken each item of data from the file and put this into an array to be spoken.

```
DIM item$(10)                          Dimension an array to 10 for the test
a=OPENIN "C:\test.txt"                 Open the test file for input
FOR j=0 TO 10
  INPUT#a,item$(j)                 Loop  to input the ten test values into the array
NEXTj
CLOSE#a                            Close the file
i=0                                Initialise the loop
REPEAT
  word$ =item$(i)                        put each item in turn into a variable and say that variable
  PRINT word$
  i=i+1
  PROCspeak(word$,Pitch%,Speed%,Voice$)        calling the speech routine.
UNTIL word$ = ""
```

The routine speaks all the words except the last one,



I will try adding a blank word at the end of the text file to see if this helps



That worked, the file needs a blank line at the end to allow the program to complete normally.

Now I need to include a routine just after the point where the current program speaks the word to obtain an input from the student and compare it with the correct spelling.

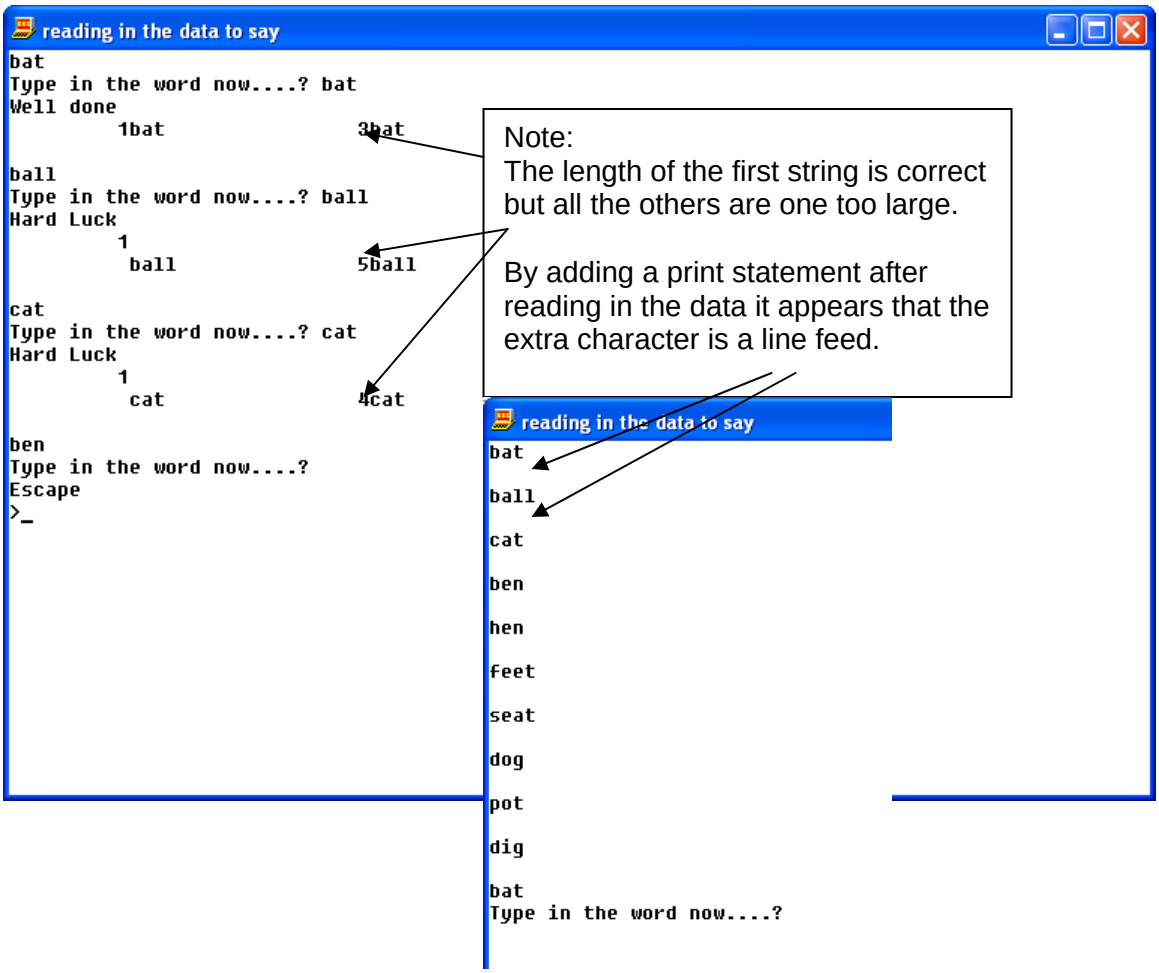
```
DIM item$(10)
correct=0
a=OPENIN "C:\test.txt"
FOR j=0 TO 10
  INPUT#a,item$(j)
NEXTj
CLOSE#a
i=0
REPEAT
  word$ =item$(i)
  PRINT word$
  i=i+1
  PROCspeak(word$,Pitch%,Speed%,Voice$)
  INPUT "Type in the word now....",student$
  IF student$=word$ THEN
    correct=correct+1
    PRINT "Well done"
  ELSE
    PRINT "Hard Luck"
  ENDIF
UNTIL word$ = ""
PRINT "You scored ";correct;" out of ";i
PROC_comexit
END
```

After adding the extra bits of code I tested the routine.
Problems:

- It asks the student to spell the last blank word so I will have to check for this
- The total was not correct

```
PROCspeak(word$,Pitch%,Speed%,Voice$)
INPUT "Type in the word now....",student$(i)
IF student$(i)=item$(i) THEN
  correct=correct+1
  PRINT "Well done"
ELSE
  PRINT "Hard Luck"
ENDIF
PRINT correct, item$(i), LEN(item$(i)),student$(i)
i=i+1
```

By adding this line to print the values of various variables in the program I was able to see that the words in the text file included an extra character.



I created a simple routine to write a data file to the C drive

```
DIM word$(10)
FOR x= 1TO 10
  INPUT "word ", word$(x)
NEXT
a=OPENOUT "C:/data.txt"
FOR i=1 TO 10
  PRINT#a,word$(i)
NEXT
CLOSE#a
```

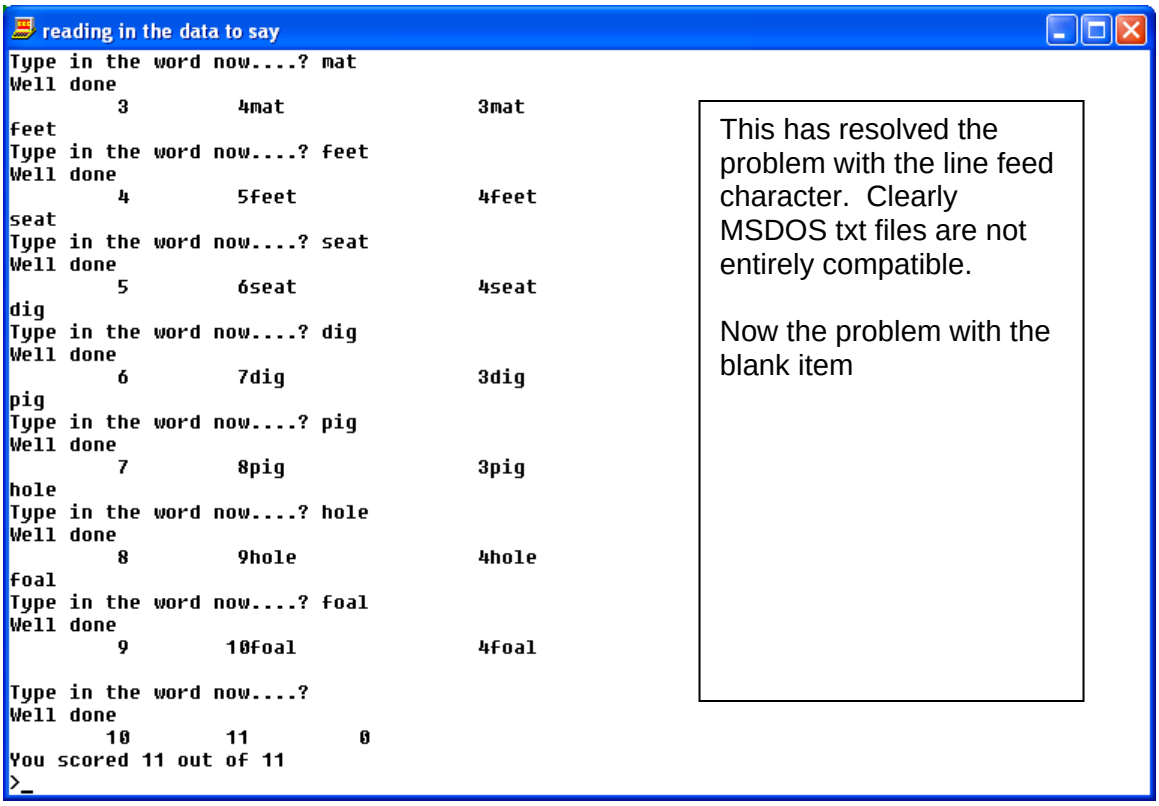
Create an array to input data into
Input 10 items of data

Open a file for output

Print the data to the file

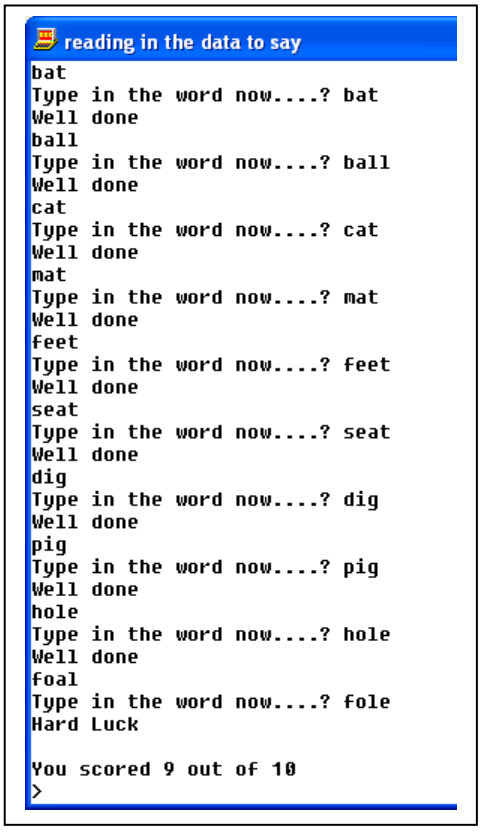
Close the file

Now I have modified my code to take data from this file rather than the one created with notepad.



This has resolved the problem with the line feed character. Clearly MSDOS txt files are not entirely compatible.

Now the problem with the blank item

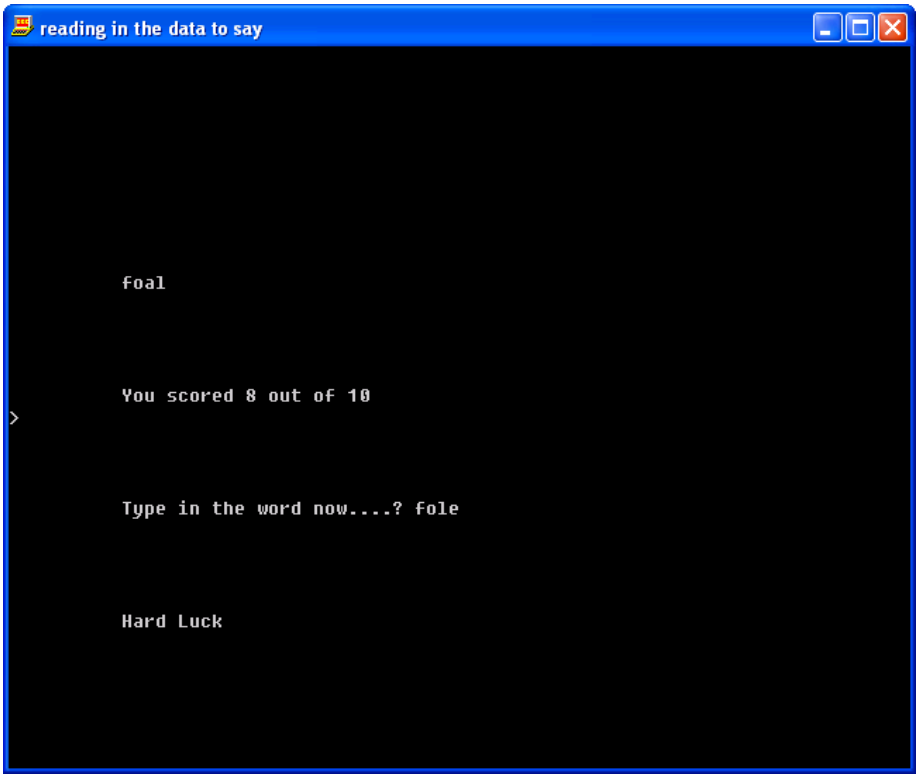



```
  DIM item$(10)
DIM student$(10)
correct=0
a=OPENIN "C:\data.txt"
FOR j=0 TO 10
  INPUT#a,item$(j)
NEXTj
CLOSE#a
i=0
REPEAT
  word$ =item$(i)
  PRINT word$
  IF word$="" THEN
    PRINT "You scored ";correct;" out of "i
  ENDIF

  PROCspeak(word$,Pitch%,Speed%,Voice$)
  IF word$<>"" THEN
    INPUT "Type in the word now....",student$(i)
    IF student$(i)=item$(i)  THEN
      correct=correct+1
      PRINT "Well done"
    ELSE
      PRINT "Hard Luck"
    ENDIF
  ENDIF
  i=i+1

UNTIL word$ = ""
```

This code now works by checking for the last blank item before outputting the final score and only printing and saying the word if it is not blank

Now I need to place the printed text at fixed locations so that it overwrites itself each time.

By using the PRINT TAB(x,y) function I was able to position the text on a graphics screen. This is MODE 0 and probably too fine for this program, I will need to ensure the spaces are cleared before writing new words on screen and experiment with the different screen modes.



Using a simple routine to blank out previous screen output before printing and saying the word, and before asking for input the screen layout now functions correctly.

```
  FOR k=0TO 15
 PRINT TAB(k+5,8) " "
NEXT k
PRINT TAB(5,8) word$
IF word$="" THEN
 PRINT TAB( 5,12) "You scored ";correct;" out of ";i
ENDIF
FOR k=0TO 14
 PRINT TAB(k+28,16) " "
NEXT k
```

This routine simply replaces what is on screen with blank spaces.

NEXT some colour has been added to the text using the colour statement. Red for hard luck, yellow for key words and well done, white for line where response is typed..

I also tried out different background colours, the grey appeared to be easiest to read.

The main body of the program now works and I need to work on choosing word lists and entering word lists. I have currently set the limit to 10 but this can easily be modified at the implementation stage.

My routine to enter and save data to a file has already been developed and the simplest solution for the desired three word lists will be to CHAIN this routine from a menu for each of the three files.

To select a word list all I need to do is to load the appropriate file from a simple menu.

This is the code I have written to use the pre written routines for the speech module.

```
MODE 19
COLOUR 136
CLS
DIM item$(10)
DIM student$(10)
correct=0
a=OPENIN "C:\data.txt"
FOR j=0 TO 10
  INPUT#a,item$(j)
NEXTj
CLOSE#a
i=0
REPEAT
  word$ =item$(i)
  FOR k=0TO 15
    PRINT TAB(k+5,8) " "
  NEXT k
  COLOUR 3
  PRINT TAB(5,8) word$
  IF word$="" THEN
    COLOUR 3
    PRINT TAB( 5,12) "You scored ";correct;" out of ";i
    PRINT TAB(3,16) "                                  "
  ENDIF
  FOR k=0TO 14
```

Set the screen mode, background colour and clear the screen.

Dimension the arrays for the word list and student input

Initialise a variable to store the current score

Open a file for input and
Input the data from file into an array

Close the file
Initialise a loop variable

Loop
Select word from list

Clear the area on the screen

Set the text colour and
Print the word on screen

If the word is blank then
Set the text colour and
Print the final score on screen

```
  PRINT TAB(k+28,16) " "
 NEXT k

 PROCspeak(word$,Pitch%,Speed%,Voice$)
 IF word$<>"" THEN
  COLOUR 7
  INPUT TAB(3,16) "Type in the word here..",student$(i)
  IF student$(i)=item$(i)  THEN
    correct=correct+1
    COLOUR 6
    PRINT TAB(5, 20) "Well done"
  ELSE
    COLOUR 1
    PRINT TAB(5,20) "Hard Luck"
  ENDIF
 ENDIF
 i=i+1

UNTIL word$ = ""

PROC_comexit
END
```

Clear the area for input

Call the speech routine with the current word
IF the word is not blank then
Set the text colour

Print instructions to type in word

Compare input to current word and update score accordingly

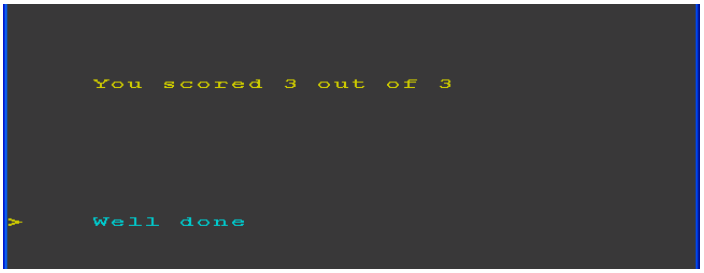
Set the text colour accordingly and print well done or hard luck

Increment counter and repeat the loop until the word selected is blank.

I have tested this routine with some sample data as I developed it. The results are shown alongside the development.

The test data had 10 items, the maximum allowed by the dimensioning (this can easily be altered) but it needs to be tested with data files containing fewer items than the maximum allowed.

I have created a data file with 3 items, cat, bat and ball.



This works without any problems

A file with non –words should be spoken but be meaningless

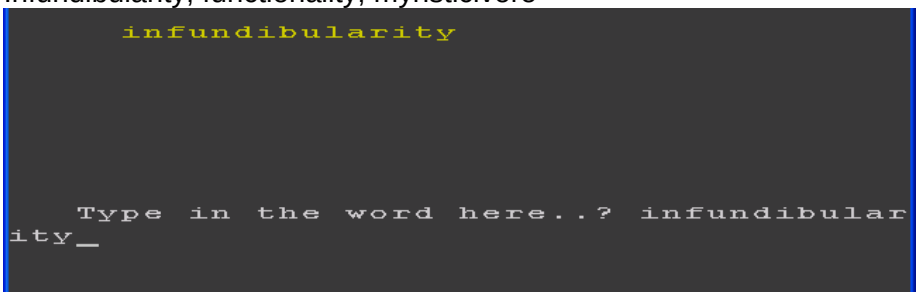
My file is ghjk, namadfad, 2e3rtg*

As suspected this causes no issue, the spoken output is just meaningless but the system functions normally.

The system is designed for primary school and short words, there is a maximum text screen size of 40x30 so words longer than 12 will wrap around.

I have used the following words to show this.

Infundibularity, functionality, myristicivore



The word wraps around when typed in but the system continues to work correctly.

The system could easily be modified if this represented any problem.

To write data to files I have created a simple program:

```
INPUT "File name " file$
filename$="C:/"+file$+".txt"
INPUT "How many words? "size
DIM word$(size)
FOR x= 1TO size
  INPUT "word ", word$(x)
NEXT
a=OPENOUT filename$
FOR i=1 TO size
  PRINT#a,word$(i)
NEXT
CLOSE#a
```

Request a name for the file
Set filename to point to text file on C drive with .txt extension

Ask for number of words
Dimension array to number of words

Get input from user

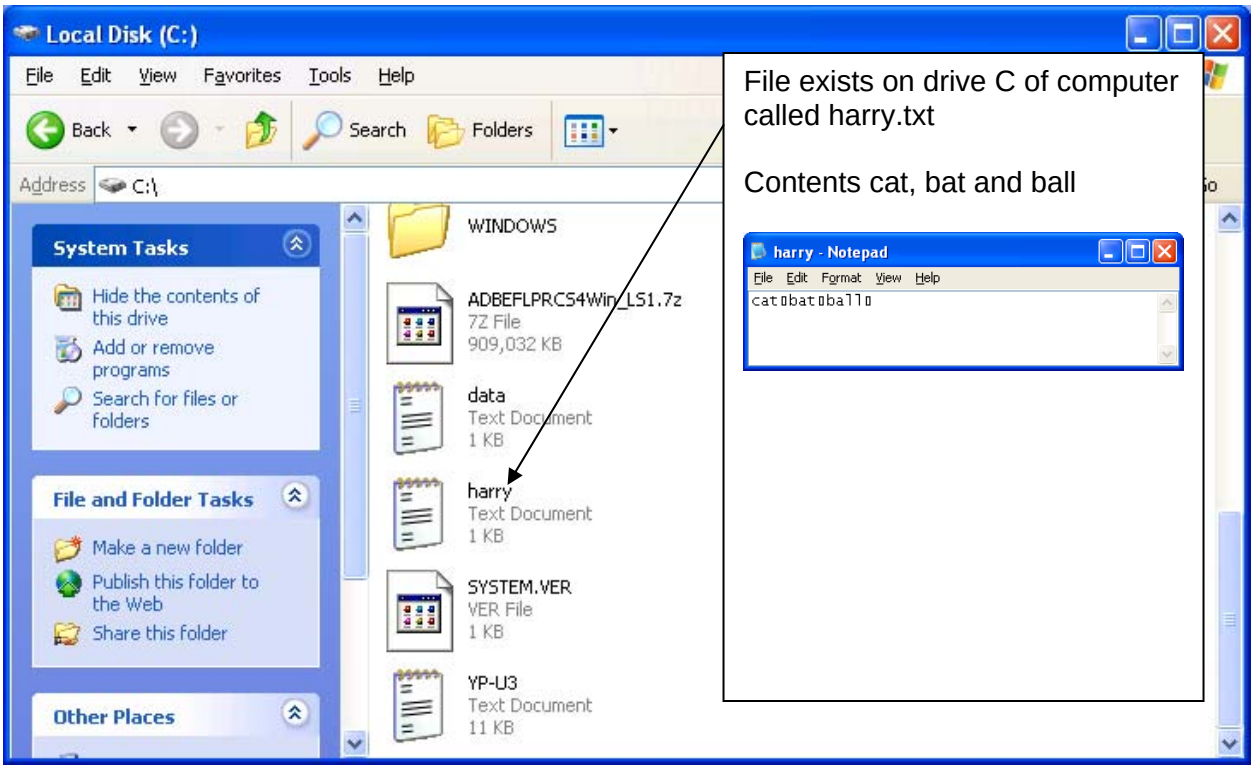
Open the file with supplied name for output

Write data to file

Close the file

This will allow the teacher to create any number of files of any size for use with the student.

Test: file name harry, 3 words, cat, bat and ball.



Now a simple modification to the start of the main program allows the user to select a data file.

```
MODE 19
COLOUR 136
CLS
PRINT TAB(3,5) "Which data file?"
INPUT TAB(5,8) file$
filename$="C:/"+file$+".txt"
CLS
DIM item$(100)
DIM student$(100)
correct=0
a=OPENIN filename$
```

Set mode, colour and text colour

Clear the screen
Ask for filename

Use data entered to create full pathname to file

Clear screen

Dimensioned to allow for large numbers of words

Open the file using the filename variable

The main program is set to store a fixed amount of data in an array. If I store the size of the data file as the first item in the data file itself I will be able to retrieve this and dimension the main program variables to the correct size.

A simple modification to the write data program allows the size to be stored as the first data item.

```
INPUT "File name " file$
    filename$="C:/"+file$+".txt"
    INPUT "How many words? "size
    DIM word$(size)
    FOR x= 1TO size
      INPUT "word ", word$(x)
    NEXT
    a=OPENOUT filename$
    PRINT#a,size
    FOR i=1 TO size
      PRINT#a,word$(i)
    NEXT
    CLOSE#a
```

Some modifications to the speak words program allows the dimensions to be set to the correct value →

```
a=OPENIN filename$
INPUT#a,size
DIM item$(size)
DIM student$(size)

FOR j=0 TO size
  INPUT#a,item$(j)
NEXTj
CLOSE#a
```

Now a simple menu system to link all the components together:

Menu options become:

1. Take test
2. Create word list
3. Quit

(Using a filename for the data file allows me to enable as many tests as the teacher wants rather than the original idea of three set tests to be modified)

```
   REM Menu
 20 MODE 19
   COLOUR 133
   CLS
   PRINT TAB(5,5) "Spelling Test"
   PRINT TAB(3,8) "1.  Take Test"
   PRINT TAB(3.10) "2.  Create a new word file"
   PRINT TAB(3.12) "3.  Quit"

   g=GET
   PRINT g

   IF g<49 OR g>51 THEN
     CLS
     PRINT "select a number between 1 and 3 please"
     GOTO 20
   ELSE
     IF g=49 THEN
       CHAIN"Reading in the data to say"
     ELSEIF g=50 THEN
        CHAIN"Writing data to file"
      ELSEIF g=51 THEN
         END
       ENDIF
      ENDIF
    ENDIF
   ENDIF
```

Set mode and colours

Clear screen

Print menu options on screen

Wait for key press and take ascii value of key pressed

If not 1,2 or 3 then clear the screen
ask for a correct value and

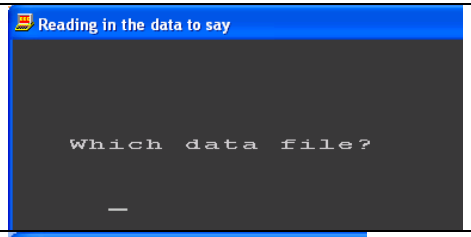
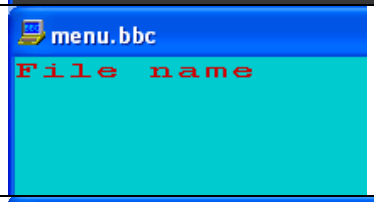
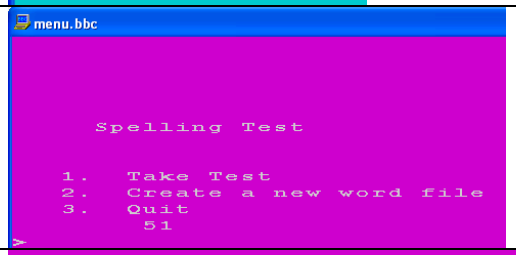
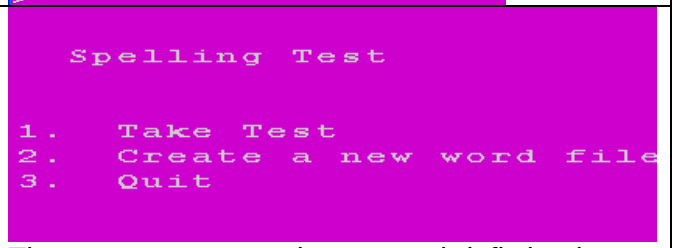
go back to the menu

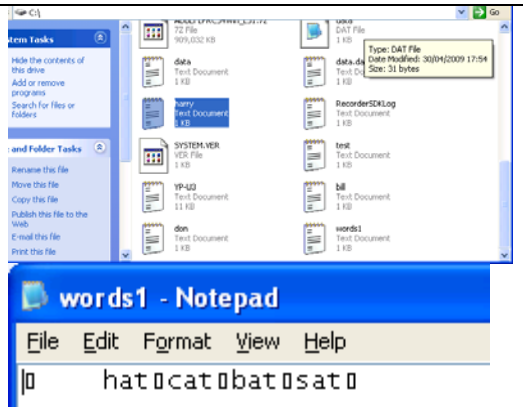
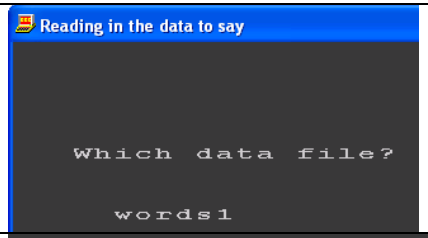
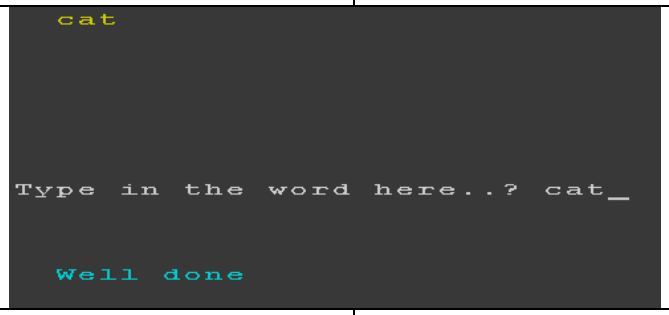
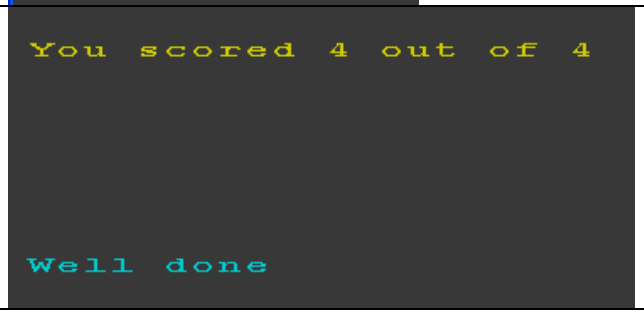
If 1 then start the take test program

If 2 then start the enter words program

If 3 then end

Test plan for full system:

Test	Data	Reason	Expected	Actual
Menu system	1	To see if take test program start	The screen will clear and it will ask me which test I want to take	
	2	To see if create new word list program starts	The screen will clear and it will ask me for a file name	
	3	To see if the system quits cleanly	The system should stop	
	4,5, -1,0 a,	Invalid data	An error message will appear and it will return to the menu	 The error message only appears briefly but it does return to the main menu. I will add a delay to show the error message.

Create a word list	Words1, 4, cat, hat, bat, sat	Valid data,	Should ask for filename, size and 4 data items and create a file called words1 on the C drive	
Take a test	Select test words1	Valid file name	Should load the test and ask me to spell 4 words	
				
	Select file noexist	File does not exist.	Should fail and give error message.	Message invalid channel and program stops. I need to add an error trapping routine to reject false file names and return to the file name request

My system design was to create a programmed solution using BBC Basic as a spelling aid for primary school children with the following features:

• A simple menu interface with 5 choices: o Three tests o Teacher option to modify the words o Quit	I have created a simple functional menu with just 3 choices. 1. Take test 2. Create Test 3. Quit The developed system has more flexibility than the original design in that it can handle any number of tests rather than just the three in the design. The create test and quit options work effectively
• The test interface keeps a score	The score in the test is displayed correctly at the end of the test
• The word may be displayed for the student to copy or not	The word is displayed, I have not included an option to turn of the displayed word.
• Simple space to type in the word	The word is typed after a prompt on screen, there is not marked space for this
• The system speaks the word using a computer voice	I found a module by R T Russell that I have used to speak the words from a file
• Differing beeps for success or failure	The system displays the words well done or hard luck instead o a beep
Additional features not in the design	
Any number of tests of any size	By using a system to create the files that allows the file name, the size and data to be input by the teacher I have made the system more flexible
Other issues	
The original brief wanted a colourful interface	I have used text and screen colours to differentiate between the three main screens and coloured text to differentiate between computer output and user input and responses.
The research indicated alphabet words and the use of images were desirable	The standard word lists were not based on the alphabet but on sounds, however, by allowing the teacher to input as many word lists as they like the need for standard word lists is avoided. The use of images does not work with many words and has been discarded as a feature

Overall the solution worked well and achieved the main goals. I have had to modify the system after testing to add an error handling routine for non existent file names and I would have liked to have spent more time on the general appearance of the solution, for example double height text and some graphic enhancements for the main program screen. I also need to include a pause to read the error message on the menu screen.

```
150 MODE 19
    COLOUR 136
    CLS
    PRINT TAB(3,5) "Which data file?"

    INPUT TAB(5,8) file$
    filename$="C:/"+file$+".txt"
    CLS
    correct=0
    a=OPENIN filename$
    IF a=0 THEN
      PRINT TAB(5,5)"Data file does not exist"
      WAIT 300
      GOTO 150
```

I have included of a label to return control to this point

A check on the file channel value to see if the file exists
Wait for 3 seconds for the user to read the error message before returning the control to the point marked by the label.

```
    IF g<49 OR g>51 THEN
      CLS
      PRINT "select a number between 1 and 3 please"
      WAIT 300
      GOTO 20
```

A wait command has been included to give the user time to read the error message

End of candidate style response

Notes from examiner

There is a basic analysis of the problem with some evidence of planned information gathering related to the initial problem. There is some research into existing solutions but this is limited - covering just a few aspects of the problem. There is a clear attempt to link the analysis to the end user requirements and this leads to useable design requirements. There is some thought given to the software requirements but no thought at all to hardware requirements.

Analysis mark:
0-3 All the items in this category are covered apart from hardware.
4-7 There is some limited evidence of planning and an attempt to link this to the problem requirements. No clear success criteria and limited analysis of existing solutions.
8-10 There is little evidence for the design requirements at this mark band apart from some limited justification for the design
Overall 4/10 for analysis

There is a clear algorithm that would provide an outline solution to the stated problem and some screen designs. There is also some mention of how the proposal solves the problem. The candidate has failed to mention how they will test the solution at this stage.

0-4 All easily covered apart from the test strategy
5-8 Good algorithm, decent designs, no test strategy
9-12 Algorithm not detailed enough nor are the designs
Overall 6/10 for the design because of the algorithm

Standard programming constructs are used effectively in the solution and there is good evidence of suitable select statements, loop structures, file handling etc. Variable names are not meaningful in many cases but at least these are not ambiguous. Standard use of i, j and k as loop indices is acceptable. Arrays are used effectively. Solutions are effective and fully functional but not always efficient. This is work bordering on 9-11 but with some inadequacies.
Overall 8/11 for use of coding features

The development is very clearly described with excellent evidence to show the development of the solution. There is a critical discussion of the process showing how the plan has been modified in light of testing during development. The commentary is clear and detailed and the overall solution is effective. There are some very minor issues and inefficiencies but no significant errors, hence 6-7 marks.
Overall 6/7 for development

There is not initial test plan but the development testing is well structured and clear, the final product testing is planned and covers various aspects of the system. There is no evidence of others testing the system and the system is only tested on the development computer. This will cause problems for this solution and these have not been identified, hence 4-7 category.
Overall 5/10 for testing

There is an evaluation of the system against the design criteria. Modifications and limitations have been identified and there is evidence that these have been dealt with.
There is no evidence of comment on group activities.
The report is well structured with good use of technical language and few errors in punctuation, grammar and spelling.

Apart from a few omissions this is top range work, but these omissions mean the top band cannot be awarded hence 4-7
Overall 7/10 for evaluation

Overall mark 36/60 - approximately C grade

We would expect this to be supported by more evidence than can be seen in the report for example: a printout of the finished program or a video showing testing in progress. We would certainly like to hear the sound or have some comment from the teacher that it does in fact work. This would be evident perhaps from end user testing evidence not supplied with this work.