

## Initial Decisions

- Choice of language. Any computer language will be able to show the candidates' ability to use the skills required. Ensure that you know how to define files. Note that if you use application software, of which Access is a very popular choice, then the candidates must define files themselves. The use of wizards and auto code will not produce work that is the candidates' own. That code cannot be considered for marking. Candidates who use wizards to define files and produce screens deprive themselves of the opportunity to gain many of the marks.
- Project topic. It is best if this is meaningful to the candidate, and relates to their school, or a business of a relative, or a club that they are interested in. It needs to involve entering data, storing data, manipulating data and outputting results. It needs to use at least one file of data; two or more files are often needed. It is rarely a good idea to choose a project that is mainly mathematical as the opportunity to use files and different data types is not usually there.
- Length. Many very good projects are about 30 pages. If the project is more than 50-60 pages it is likely that a candidate is spending too much time on this subject.
- Simplicity. Do not make the project over-complicated.
- Objective. Bear in mind that the main objective of this project is to allow the candidate to show how well they can write computer code and how well they can document that code.
- Intention. This document is intended to give ideas on how to arrive at a project topic, some examples of the type of presentation suggested, and some guidance on how to satisfy the criteria for marking this project. It is not intended to be a complete example of a typical Paper 2 project.

## Defining the problem

This needs to be a short introduction, perhaps two paragraphs long, that gives the context of the problem and exactly what the program will solve.

It might indicate the sort of data available, and the sort of output that the user will want.

## Design

The candidates need to show how the structure of the code was thought out. This can be done using some flow charts, data flow diagrams, pseudocode or structure diagrams.

The file(s) need to be designed, so the candidates need to show the structure of their records, with data types, size, if restricted, and any validation rules. If the decisions are out of the usual, the reasons for such a decision should be given.

If the relationship between files needs it, an E-R diagram might be appropriate. Input and output screens may need to be designed.

The screens and record structures should be the candidates' own diagrams, not application software print outs. The project should not be a chance for candidates to show their ability to use such software, typically Access.

It should be possible to see the connection between the design and the code that is written.

The procedures in the code should correspond to the steps in the flow charts. The record design should correspond to what is in the code.

## Code

All the elements that are suggested to make the code as readable as possible are in the syllabus (pages 22-24).

The aim should be that the code is easily understandable, not just to an expert in the language being used, but to anyone looking at it. This is achieved partly through layout, partly through using meaningful names for the variables and procedures, but mainly through annotation.

Annotation is the use of comments after many of the lines of code or in the heading of a procedure or function. A page of tightly written code with no annotation is almost meaningless as whilst it can be understood eventually by someone with knowledge of the language, it usually conveys no meaning at all.

As an example:

```
Function cube(number:integer):integer; {this function evaluates the cube of a whole number}
Begin
    cube:=number*number*number {calculates the cube}
end
```

It would also be good to add a hand-written note that this function can be found on page *n* of the design.

Or the contrast here.

```
for i=1 to len_
for j=1 to len_
array(i,j)=mid$(txt,((i-1)*len_)+j,1)
next j
next i
```

```
for width = 1 to len_           'starting a loop within the array
    for height = 1 to len_      'starting inner loop
        array(width, height) = mid$(txt,((width-1)*len_)+height,1)
        'setting all the characters in the array
    Next height
Next width
```

This is a good example:

```
Private Sub cmdnewbook_Click()  
    ***Declare a variable for storing the book number  
  
Dim booknumber As Long  
  
    *** determine the next available book number by...  
  
    *** ...1 moving the record pointer to the last record  
  
Adobook.Recordset.MoveLast  
  
    *** ...2 storing the book number of the last book  
  
Booknumber = Abobook.Recordset.Fields("BookNumber")  
  
    *** ...3 Calculating the next available book number  
  
Booknumbe = booknumber + 1  
  
    ***Add a new blank record using the AddNew method  
  
Adobook.Recordset.Addnew  
  
    ***Fill in the next available book number  
  
Txtbooknumber.Text = booknumber  
  
End Sub
```

PLUS, by the side and handwritten, a note as to where this module is in the design.

Some candidates highlight their examples of the 8 specific programming skills asked for. Others print out separately these examples to show that they have been used.

## Testing

It is usually clearer to see what testing has been done if the candidates draw up a table to show what they tested, the expected outcome, the actual outcome and where the appropriate screen shot is to be found.

Only tests with screen shot evidence are relevant to the marking.

Candidates are asked for at least 8 tests.

Many candidates show large scale testing of data entry and validation testing.

It is good to see *some* examples that show that this aspect of the project works, but the essential is to produce a series of tests that show that the code that was written achieves its objectives, and works as the programmer wants.

This may well involve testing subprograms or procedures along the way.

There are, of course, an infinite number of pieces of data that could be used to show that a program works in all conditions.

The candidates are being asked to select sufficient tests to show that their program does what it is expected to do without necessarily being exhaustive.

### **Documentation**

Program Documentation is the complete description of the software, intended for use when altering or adapting the software. It usually includes a statement on the purpose of the software, the language used, the hardware it was written for, when it was written, any restrictions on the use of the software, the format for input data, examples of output, flowcharts, record and file structures, program listings and any additional notes.

**Many of these will already be in the design section and can be referenced by the candidate without re-producing the diagram or code.**

**Note: If candidates follow the systems project as set out in Paper 4, it is very difficult to get a pass in Paper 2, as there is not enough program documentation in the systems project, Paper 4.**