Multiple Clioice Questions Test Scoring and Item Analysis Tool

By: Mohamad Zamberi Saad

University of Nottingham

Submitted September 1995, in partial fulfiilment of the conditions for the award of the degree of Msc in Information Technology

Supervisor: Mrs. Jean Hitchings

ABSTRACT

This project has successfully developed a computer program that can be used on IBM-compatible personal computers to do multiple-choice questions test scoring and item analysis. The program is menu driven, with pull-down menus act as the main interface with the user. The program reads input data from an ASCII text file, which filename is given by the user. The user has to prepare the data file before using this program. The output given is the scores obtained from the test and the item analysis results, which consists of discrimination index, difficulty index and response count. The output displayed on screen can also be saved to a text file. The user can make several options in giving commands for the program to score and analyse the item. These include scoring using guessing correction and displaying response count in percentage.

TABLE OF CONTENTS

TABLE OF FIGURES

Chapter 6

CHAPTER 1

INTRODUCTION

1 .1 Introduction to This Project

This project is concerns with helping educators by providing an easy and cheap way of scoring and analysing MCQ test questions. The method of providing the help is by developing and completing a computer program that can be used as a tool to score and analyse MCQ test items. The computer program to be developed must be easily accessible by the educators, easy to use and cheap. Educators that will benefit most from this project are expected to be the educators that are teaching in lower-level institutions, like primary and secondary school teachers. Higher-level educators in higher institutions like universities may be can access and use the program that has been developed specially for their university. However, they can still use the program developed through this project if they like to.

1.2 The Needs to Do This Project

There are several reasons why this project topic has been selected. While some of them are academic reasons, some of them come from my personal considerations. Education has been selected as the field of research because of my personal consideration. As an ex-teacher, I realise the heavy burdens that teachers have. It is not unusual they have to do so many works in a very short time. Therefore, a topic that can benefit teachers and lighten their burdens is more preferable.

Testing field is chosen (and not Computer Assisted Learning or time-tabling, for example) because assessment and testing appeared to be one of the most difficult division in education.

Furthermore, while programs like CAL, nre growing very fast in number, there is not much attention given by software developers to help teachers in speeding up test scoring and item analysing, except in managing and processing already obtained scores. So, programs like the one to be developed in this project need to be produced more.

MCQ is chosen because its scoring procedure is more direct compared to other kinds of questions. For example, the 'fill m-the-blank' type will need some expertise value in the program that evaluate the answers especially in determining the score from a wrongly spelled answer. Any research in such area will need much more time to be completed. So, MCQ is chosen because of the time constraint that does not permit too advance research to be completed.

There are several programs that are able to do what is intended to be done in this project. One of the programs that deals with computer automated marking is ceilidh system (Computer Environment For Interactive Learning In Diverse Habitats), developed and currently running at the University of Nottingham. In addition to managing programming skills assessment, ceilidh also have a facility to do MCQ marking. This facility can also be used to give questions to students, and the students must answer the questions while using a computer that is connected to the network server that provide the ceilidh service.

While programs like ceilidh are already exist, still there are the needs to proceed with this project. The currently exist programs are normally inaccessible by the school teachers. To use the programs, not only the teachers have to consult the university that built the program, but also they will need adequate skills to use computers in the same platforms where the programs have been built for. Furthermore, these programs are normally difficult to be transferred from one platform to another. For instance, if ceilidh were meant to worked on UNIX-based machines, it will become almost inaccessible to the teachers who only has got training to use DOS-based personal computers. So, a small program that will be easily accessible by the teachers and will serve the purpose of scoring and analysing MCQ test items still needs to be developed.

The main purpose of this project is to provide the teachers a cheap and easy way to release their heavy burdens of scoring and analysing tests' answers. The teachers should not have to spend so much money to buy new machines or devices. This can be achieved by providing them with a computer program that can fit into a diskette and can be run using their existing personal computer at school or at their home.

1.3 The Scope of This Project

This project should be completed in two and a half months. So, it is suitable for this project to include only the first live phases from the usual seven phases in a system life cycle. The phases to be included are:

1. Identifying problems and objectives.
2. Finding the information required.
3. Doing the system analysis.
4. Designing the system.
5. Developing and documenting the software.

In this project, phases 1 and 2 are reported in chapters 2 and 3. Phase 3, the system analysis, is shown in chapter 4. The software development (coding) is explained in chapter 5 and lastly, the software documentation is located in chapter 6. The two phases that are not included in this project due to the time constraint are:

1. Testing and maintaining the software.
2. Implementing and evaluating the software.

Testing, implementing and evaluating the software will consume so much time that implementing it will most likely to make this project exceed the time limit. So, it has been decided that this project will only do the first five phases.

CHAPTER 2

INTRODUCTION TO THE PROGRAM

2.1 Introduction to The Program

2.1.1 What This Program Is

The program described in this report is entitled "Multiple Choice Question (MCQ) Test Scoring and Item Analysis Tool". Like suggested by its title, it can be used as a tool to accomplish the task of scoring MCQ tests and analysing the functionality and usability of each question in such tests.

This program helps the users by reducing the time spent in scoring and analysing the test items (in educational tests, a question is also known as an item). Instead of having to mark and analyse the test manually using paper-based works, the users only have to ready an ASCII text file by any text editor to be used as the input data to this program. The text file should contain all the answers of the test. The program will read the text file and then do the works of giving scores to the students and analysing the test items.

While using this program, the user can change several options regarding the scoring and analysing procedure. These include the way scoring is done and the way to report the results. The result of this program is the test score for each student, and the analysis results concerning each question in that test. The output is in two forms, which are on-screen form and in-file form. The user can browse through the results displayed on the screen, and can then write the result to a disk file for further processing purposes, such as editing it in a word-processor and printing it

2.1.2 Requirements

This program is written and compiled using Turbo Ctt version 1.01 for DOS. The user must run this program from an IBM compatible personal computer (PC) with at least 256kb RAM, operating on at least DOS operating system version 4. The PC used must also use a colour monitor, capable of displaying text in 16 colours mode.

2.1.3 Potential Users

Most program developed have the potential users in the mind of their developers. This MCQ test tool program is intended to help people who deal with MCQ tests, regardless of the level of the tests. So the potential users of this program include all educators and test organisers, such as teachers and lecturers. A teacher in a primary school may be using this program in more basic settings mode, where the scoring scheme does no correction to the raw marks. Educators at higher level such as lecturers in universities can benefit from more advanced features of this program, for instance using guessing correction scheme in scoring a test. A test organisor that develops tests periodically and used some questions repeatedly, for example admission tests, can take most advantage in the analysis part of this program. He/she can find out the functionality and usability of an item before deciding whether to use the item or such items in future tests.

2.2 Motivations for Making this Program

In every system developed, there must be some motivations that stimulated the idea of developing the system and several factors that encouraged its development. For this MCQ test tool program, there are two major motivations that encouraged its development. The first one is the consideration of the time that can be saved by using this program, including the advantages of using computers to do recalculation in much better ways. Secondly, we are hoping that the provision of a cheaper alternative compared to much more expensive tools currently used by specialised test organiser will contribute to the extensive use of computers among educators in schools. It is on these concerns that this program has been built.

2.2.1 Time Saving

Time saving is the first and biggest motivation to the building of this program. As already acknowledged by most test administrators, not only the task of making a good MCQ test itself is very hard and time consuming, but also the task of scoring, and then analysing the functionality and the usability of each question in the test. The scoring process involves scanning all the candidate answers, each time comparing them with the actual answers and then giving the mark to that item. The number of comparisons involved is the product of the number of students and the number of questions. For instance, if a test seated by m students contains n questions, the marker must do m x n comparisons to complete the scoring process. This is not including the process of giving scores for the test as a whole to the students, which involves the counting of right answers. The process will consume much more time if guessing correction is involved, where the number of wrong answers also had to be counted and be put into a formula to obtain the corrected score.

The process of analysing each test item is more tedious compared to the process of scoring the test. This involves scanning all the candidates' answers sheets for each item. This tediousness can be seen by taking response count as the simplest example. To count the number of responses for each choices in an item-, the marker has to examine the answer sheet back and forth, one choice at a time. For example, to count the number of choice "A", he/she has to go through all the answer sheets to count the occurrence of "A" for that particular item. If the question has 5 choices, i.e. "A" to "E", he/she has to go through the answer sheets 5 times for that item alone. So there is considerably much time needed to complete the item analysis process.

If a computer program can be used to do all these tasks in one instance, the time saved will be significantly important. The time saved can then be used by the educators to make better decisions for next test preparations. Although the educators have always managed to come up with the student scores, it is very discouraging for them to come up with the item analysis results, due to the time taken to complete the process. If they do not do item analysis, they will lose much useful information about their own teaching methods and the item's capability. These kinds of information are very helpful in making judgement about previous teaching methods used before the test, and can be used to determine the reliability of each item. In conclusion, a program that can speed up the process of scoring and analysing the test items can save educators so much time, which will give them more time to make better judgement on the test and their teaching methods.

2.2.1 .1 Encouraging Continuous Student's Development Diagnostics

The development of this program is also based on the idea of providing a facility that is not only fast, but also "clean" in scoring and analysing the test items, which in turn will encourage the continuous monitoring of students' progress, compared to paper-based systems. Here, the word "clean" means that the report given is flexible enough to change with possible changes to the answers given, without cluttering the report. For example, if a wrong actual answer is given and then changed; this program will also taking the new input as the usual input, re-scoring and analysing the test. This will save much time, again. In current practice, a recalculation must be done to all the scores and item analysis that has involved with a wrongly given answer item. In contrast, re-calculations using computers are just the matter of running the program again, and the new output will be given at once. By providing a program that will give clean and accurate reports about each test item, we will provide the educators a facility that can be used to diagnose any flaw in the instructions given while teaching the students before the test. For instance, if an item is reported by this program to have a zero difficulty index (that is nobody answered it right), may be the given actual answer is wrong, or the educators can suspect that some major concept misunderstandings have been contracted by the students. If the answer given is wrong, he/she only have to run the program again. If there is evidence that misconceptions had happened, he can correct that in the next meeting with his/her students. In other words, by providing a program that will help the educators in analysing the test items, we can help them in giving better service to their students, by both preparing more reliable tests to evaluate them and by correcting any misconceptions in their students mind. Both the educators and the students can benefit from this kind of programs.

2.2.2 Encouraging the Use of Computers in Schools

Another concern that arouses the idea of building this program is to provide a cheaper alternative for MCQ test processing, which in turn will encourage the more extensive use of computers in schools: The uses of computers in primary and secondary schools are currently not very impressive compared to sectors as industrial, business or in higher educational institutions. This is due to the factors such as the high cost, the small number of computer software that is appropriate for school administration purposes and the lack of knowledge and consciousness towards the computer use among school administrators and teachers (Ray, 1991). Though the cost of buying a unit of computer is quite high for a school, many schools have at least one unit of computer located in the administration office to do at least clerical works. So the cost factor is not a major factor that prevents the educators in schools to use computers. If appropriate software can be, developed and its functionality can be showed to the administrators, perhaps their consciousness will be greatly improved. One such software is the program that can help them mark the multiple choice question tests and giving help in interpreting the tests.

Currently, there are several kinds of machines used in scanning the candidate answers, recording them and then scoring and analysing the items used. These machines are usually expensive enough that only certain companies which are specialising in test processing are using them. Usually, an average school cannot afford to buy those machines. By giving schools the access to this program, we are actually giving them an alternative to do MCQ processing in a cheaper way, since this program can be run through a typical PC.

In conclusion, two motivations have encouraged the development of this program. The first one is the consideration of the time that can be saved by using a computer program to replace paper-based works. By giving results in a very short processing time, we are hoping that the educators will not be discouraged to do item analysis anymore. The second one is the idea of providing a cheaper MCQ test tool, which is hoped to contribute to more comprehensive use of computers in schools.

CHAPTER3

THETHEORYANDPRACTICEOF MCQ TESTS SCORING AND ITEM ANALYSIS

3.1 Introduction

MCQ tests have got more critics than any other forms of testing do (Wood,1991,p.32). However, MCQ tests are still being widely used. They are used in teacher-constructed tests in schools to assess the students and instruction. In higher level admission tests, MCQ tests are used to select only the best-performing candidates. It is also commonly used in medical education, where the factual domain to be tested is usually very big.

The main critic to MCQ tests is that they failed to measure higher-level cognitive outcomes, as done by essay questions. However, we should not expect MCQ tests to do what they are not particularly designed for. As Wood (1991,p.32) suggested, MCQ can be justified by insisting that it is a technique that does a particular job and not more than that. The job MCQ is doing is making students to read the questions and then think, not writing. We should not expect MCQ to do more than that. As far as its job is concerned, MCQ is doing it very well. So, according to Wood, it is not that MCQ cannot do its job, but the critics wanted MCQ to do the job that is suitable to be accomplished by other methods.

The fact that MCQ tests are still being widely used suggests that it has several advantages. One advantage that MCQ tests have over essay tests is that MCQ tests can sample the content of certain domain extensively (Wiersma,1990,p.43). If a teacher wants to measure the outcome of his/her instructions in a very big area, he/she can do so by assigning an MCQ test containing enough questions. He/she cannot do that with essay test because of the test time limit. This is true since in MCQ tests, to indicate the answer, a candidate is expected to just mark the answer on the answer sheet. While answering essay questions, the students spend more time in organising his/her essay and writing it down

Besides being able to extensively sample a domain, MCQ tests also take less time to be scored compared to essay tests. Very simple and objective scoring methods are used, which will also ensure that there should be no bias in the scoring process, a phenomenon proven to happened in essay tests scoring. This is one of the reasons why big assessment organisations that prepare the admission test use MCQ tests to select competent candidates.

3.2 Scoring

There are several scoring techniques used in MCQ tests. These include counting the raw number of correct answers and applying guessing correction The most widely used is the counting of correct answers, which is very popular in teacher-constructed tests at school level. The next popular method is applying guessing correction to the number of correct responses, which is considerably common in very big admission tests and medical education tests. Other variations of scoring methods include dual response and the use of differential weightings of responses. However, since these variations are not very common, they are not included in this research

3.2.1 Number of Correct Answers

The most extensive use of MCQ test scoring, due to its simplicity, is by counting number of correct answers. No alteration is done to the score obtained. The advantage to this technique is obvious, that it is very easy to use and understand. The disadvantage is it does not cope with random guessing that might have been adopted by the candidates during the test.

3.2.2 Guessing Correction

As the name suggests, guessing correction is done **to correct the** supposedly lower scores obtained by a candidate, which has become higher because blind guessing were practiced through out the test. To understand how to correct the score, one must understand how the guessing process might improve the candidate's *score.* For an example, take an MCQ test which use 5 choices in each question. If a candidate use blind guessing to *answer* a question, the probability of he/she to hit the right answer is 1 out of 5, or 0.2. If he/she guesses on 20 questions, the probability is that he/she will get 20 / 5 = **4** questions right. Guessing correction aimed to eliminate the points gained by blind guessing. The general formula for calculating the actual score is:

score = right - (wrong / (n - 1)) ,

where n is the number of choices in a question. For example, take a test that contains 100 questions of 5 choices. If a student answered 70 questions and omitted 30, and get 54 right and 16 wrong answers, the corrected score will be:

score = **54 - (16 / (5 - 1)) = 54 - 4 = 50 .**

What is reflected is that from 54 right answers, the candidate might have been certain on only 50 of them, and guess blindly on 20 of them, which he/she hit 4 questions correct. By deducting the possible points by random guessing, guessing correction is trying to eliminate the extra points that came from blind guessing.

3.3 Item Analysis

Item analysis is done to gain useful information that will help in 'making decisions about the students, the teaching method and the item itself **(Wiersma,1990,p.240-241).** The error pattern showed by a student on several related items can lead to decisions about that student. The error pattetn showed on a group of collective items aimed to measure an objective can support decisions about the pace of the instruction. Performance patterns on an item itself can show whether that item is adequate or not.

The most common and traditionally used analysis processes are the difficulty index and discrimination index (Wiersma,1990,p.143). Lennox (1974,p.21), when discussing the analysis required to make up a test paper, suggested that it should be no more than seven figures to be presented. These are difficulty index, discrimination index and five response count results, one count for each possible answer. Lennox is not in favour of using too complicated mathematical techniques, because it can bring the dangers of ignoring the item contents due to the extensive favours for their calculated indices and the possibility to make the figures very hard to understand. Due to these reasons, this research will only concentrate on those three analyses, that are difficulty index, discrimination index and response count.

3.3.1 Difficulty Index

The difficulty index of an item shows the percentage of students who got the right answer for that item. The formula to calculate this is very simple:

difficulty index = r / t ,

where r is the number of correct responses

and t is the total number of candidates.

This formula will give a reverse definition of the difficulty index, where the larger the index, the easier the item That is why some people call it the easiness index, like Lennox (1974) did. The value range is from 0 to 1, with 0 means that no candidate answered it right and 1 means all candidates got it right. So, a lower difficulty index means that most of the candidates cannot answer the item correctly.

The simplest use of the difficulty index is in determining whether candidates could have grabbed a concept tested in a question and in certifying the suitability of an item itself. If the difficulty index on an item that meant to test what have been taught in class is very low, this means that the students as average still did not get the concept. It is also used to detect the fault of giving too many clues in a question. For example, if a difficulty index for an item is expected to be low, but it turns out to be high, this may be because the item has given so much hints that most candidates getting it right.

3.3.2 Discrimination Index

Theoretically, the discrimination index is the quantity of item contribution to the final order of tbe candidates, based on their total mark (Lennox,1974,p.21). In other words, discrimination index is the degree of effectiveness of an item in splitting candidates with high scores on the total test from those with low scores (Wiersma,1990,p.145). The value range for the discrimination index is between 1 and -1. When all the candidates who get an item right stand higher in the order than all those who get it wrong, the item is said to have discrimination index of 1. There, that particular itemisgoodind iscriminating higher-scorers and lower-scorers in that test as a whole. In contrast, if the value of discrimination index is -1, that item has totally failed to discriminate between high achievers and low achievers of the test. Such items are bad, and should be eliminated from the next test. The value of 0 means that the item is also fails to discriminate students, but it is better than a negative value. The formula used to calculate the discrimination index is:

discrimination index = $pH - pL$,

where pH = difficulty index for the higher group

and pL = difficulty index for the lower group.

The number of candidates in a group is 30% of the total number. So, to calculate the discrimination index, one must first marks the answers, sorts the answer sheets according to position and then choose 30% of highest score candidates and 30% of lowest score candidates. For each item, he/she must count the correct answers of each group to get the pH and pL.

The use of discrimination index is mainly to measure how good an item in separating good and unsatisfactory candidates. Often, this index is used in very high quality admission tests, to make sure that the tests themselves are capable of filtering poor candidates from good candidates. For in-class purpose, discrimination index alone is not a decisive factor in evaluating the item effectiveness. Usually, it is always used in conjunction with difficulty index and response counts to detect any flaw in the test items or in the instruction given.

3.3.3 Response Count

Response count is done in a very direct way, that is counting the occurrence of each choices in an item The total number of responses counted, added with the number of omissions, ought to be the total number of candidates. The primary use of response count is in studying the students' error patterns. As Gronlund (198 1,p. 187) stated:

"The nature of the incorrect alternatives selected by pupils provides clues to factual errors and misunderstandings that need correction."

For example, if a wrong choice is constantly being chosen by all candidates in an item assured to have no flaw, it is very likely that the students have seized a wrong notion.

Response count can also help in evaluating the effectiveness of distractors. For example, if candidates are in favour of only two out of five choices, the other 3 alternatives are not very good distractors, because the candidates know that those 3 are the impossible choices. In that case, what is supposed to be a 5 alternatives question has turned out to be a true-false item

3.4 The Problems with Manual Scoring and Analysis

The main problem with most statistics works is the time consumed in processing the data This is also true with MCQ tests scoring and item analysis, where one must spend a lot of time in counting the answers of the candidates.

3.4.1 Scoring Problems

Scoring MCQ tests is a process which is fully exposed to errors. As Gronlund(1981,p.292) stated:

"Despite the simplicity of hand-scoring procedures, the task is fraught with possibilities for error."

In scoring a candidate's answer sheet, one has to compare all the candidates' answers with actual answers as many times as the total number of questions. If the number of questions is small enough he/she can simultaneously count the number of matching comparisons while comparing them That will be difficult to accomplish if the test contains so many questions. So, in a big test, while comparing answers, one must first marks the candidate answer sheet with one recognisable mark on each right answer. After finishing the marking process, he/she must then count the number of correct answers to get the score for that candidate. If guessing correction is to be used, one must also count for the total number of wrong answers. Then, using both numbers, the corrected score can be calculated. All these are really exposed to counting error and they are very time consuming.

3.4.2 Item Analysis Problems

The success of doing item analysis lay on the success of counting answers. All three analysis results are based on the counting of answers. The main problem of human counting is the exposure to mistakes. In item analysis, re-counting has to be done if there is a counting mistake. An example where the counting has to be repeated is in the response count process. If the total number of answers on each alternative, added to the number of omissions is different from the total number of candidates, then there must be a mistake, and the counting process has to be repeated. Here, accurate figures have to be gained because not only it will defect the response count result, but also the difficulty index that will be calculated. Difficulty index can be directly affected by the response count result, since one can just take the number of correct answers in the response count result to calculate the difficulty index.

While the difficulty index can benefit from overall response counts, the discrimination index cannot. Discrimination index can only be done after the scoring process is completed and the answer sheets have been sorted according to marks obtained. Only 54% of the answer sheets will be used, that are the top 30% and the lowest 30% of all answer sheets. Here, the counting process is started again to get the number of correct answers on each group to calculate the pH and pL. Again, time-consuming and error-prone contribute to the problems associated with finding an item analysis result.

3.5 Alternatives to Manual System

This research is motivated by the idea of reducing the problems in scoring and analysing the items in MCQ tests. As has been previously discussed, time-consuming and error-prone are the major problems with the processes. An alternative to doing hand-scoring is by using machine-scoring, which require specially designed answer sheets. However, as Gronlund(1981,p.246) implied, the machine-scoring should be used only if the number of papers to be scored is big enough so that the expense is worthy. Besides, the companies who offer machine-scoring facility are not always easily accessible to all schools. So, using machine-scoring is not a good alternative for teachers.

If the use of computers is the primary concern, we can suggest the use of a statistical package to automate the manual system. However, statistical packages is not a good choice because of two masons. Firstly, if the intention is just to do scoring and item analysis, statistical packages are usually not worth buying because they are expensive. Secondly, statistical packages are designed to be used by more serious statisticians, working on more advance numerical computations. The users usually need some training in statistics and the understanding of the package before they can really benefit from the package. So, generally, statistical packages are quite difficult to be mastered by average school teachers and will increase the unsuitability of using them to do the scoring and item analysis.

One alternative that is more suitable is a computer program that has the following characteristics. It must be cheap, simple, not very big, can be operated in typical PCs currently used in schools and can be easily understood and mastered by those who have already know the item analysis methods. This research is intended to prepare that program

Chapter 4

System Analysis and Design

4.1 Introduction

In a system life cycle, there are at least 7 phases involved. The first phase requires the system developer to identify the problems occurred within the current practice, to find any chance to improve the situation and to discover the objectives of the project. The second phase is to determine the information requirements. In this phase, the analysts have to know what information is needed by the potential system users to perform their jobs. These 2 phases have been discussed in chapters 1, 2 and 3. The next 2 phases are system analysis and design, which will be the subject of this chapter.

4.2 Project Objectives

As the result of the first and second phases of this project, which was discussed in previous chapters, the suitable objective of this research is to develop a computer program that can be used as a tool by all educators to:

1. Quickly and accurately mark and score the student answers in multiple choice question tests.
2. Quickly and accurately calculate the discrimination index, difficulty index and the response counts of multiple choice question test items, by which can then be used by the users in validating the test itself.

It is hoped that this program will then encourage the teachers to do item analysis in most MCQ tests they develop and can then be used in improving their teaching methods and future MCQ tests they will develop.

4.3 System Analysis

The analysis phase is done to reveal the system needs. To help the process of requirements determination, this project uses a technique called Data Flow Diagrams (DFD). It is from these DFDs that further progress will be made.

4.3.1 Data Flow Diagrams

DFDs are the graphical representations of the data flows and data transformations between processes in the system. This project will use DFDs in analysing the system so that the nature of data flows in the system will be clearly seen, making it easier to move to the implementation phase.

4.3.1.1 Context diagram

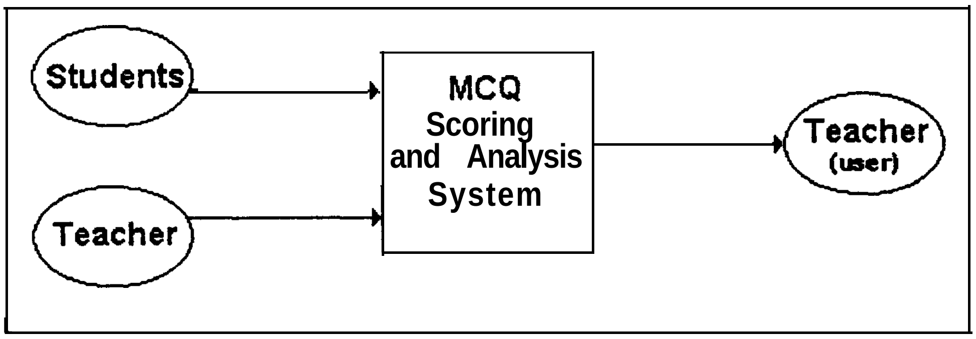


Figure *4.1 Context diagram of the proposed system*

In this context diagram, there are 2 external entities; ‘students’ and ‘teacher’. ‘Students’ will answer the test questions, providing the input to the system. The ‘teacher’ will provide the actual answers to be used by the system in scoring and analysing the test items. The output of this system will be sent back to the 'teacher', to enable him/her interpreting the outcome and the validity of the test items.

4.3.1.2 Diagram 0, or Top-level DFD

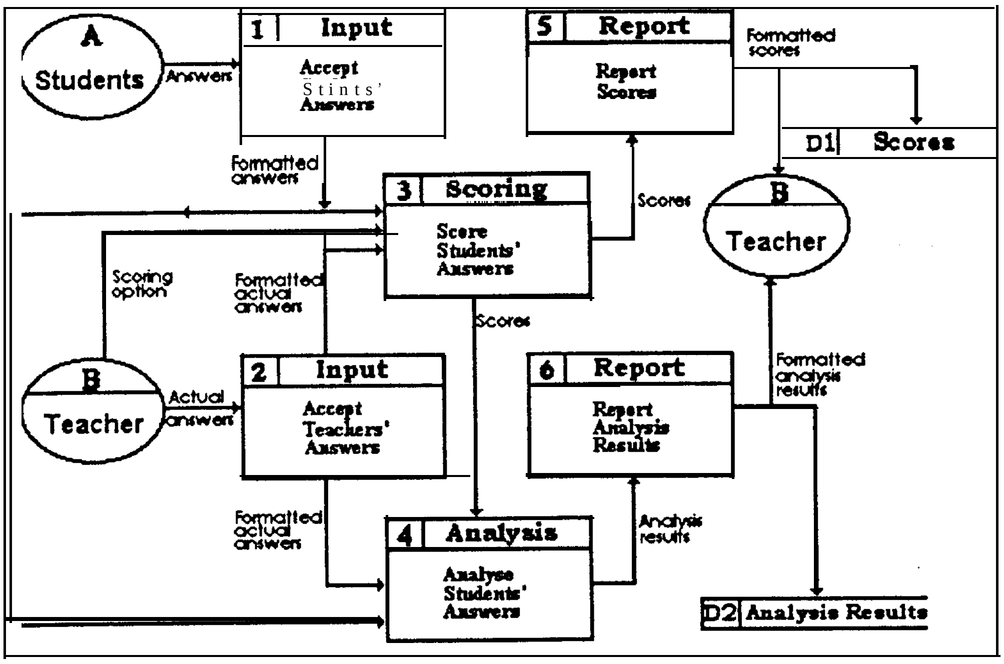


Figure 4.2 ***Diagram 0 or top-level DFD***

The next DFD developed after context diagram is called 'diagram 0' or 'level 1 DFD'. This diagram shows all the processes, the data flow and the data store in the most general form after context diagram. The details of each process in this diagram will be discussed in the next sections, with the processes in this diagram being expanded into more detailed diagrams.

4.3.1.3 Level 2 DFDs

Processes 1, 2, 3 and 4 on diagram 0 have been exploded to create more detailed child diagrams. These diagrams are called level 2 processes.

Process 1

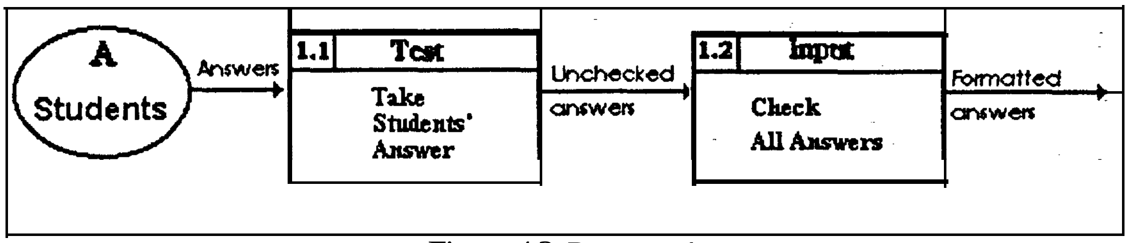


Figure 4.3 ***Process 1***

Process 1 is concerned with taking the students' answers and ensuring that the data is in the correct format. The 'students' external entity will provide the answers that will be used in scoring and analysing the MCQ test items. Before proceeding with those processes, it is vital that the data do not contain any faulty format. This will ensure that the section that will score and analyse the answers will receive 'clean' answers. It will prevent those parts from having to deal with incorrect answer formats, which will waste their time in filtering the wrongly formatted data. Child process 1.1 will take the answers from the external entity 'students' and provide the answers to process 1.2, which will in turn check the validity of the answers. The output from process 1 is the valid, well-formatted answers ready to be used by other processes.

Process 2

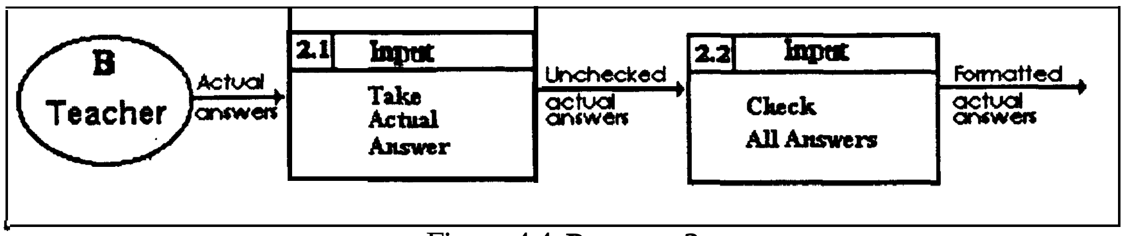


Figure 4.4 ***Process 2***

Process 2 is so much like process 1, in the sense that they both take the unformatted answers, checked the answers' validity and then give all valid answers as output. They only differ in the amount of data they have to deal with. While process 1 has to take and check the answers of all students, process 2 only has to deal with a set of answers, i.e. the actual answers given by the teacher. So, their DFDs look so much alike.

Process 3

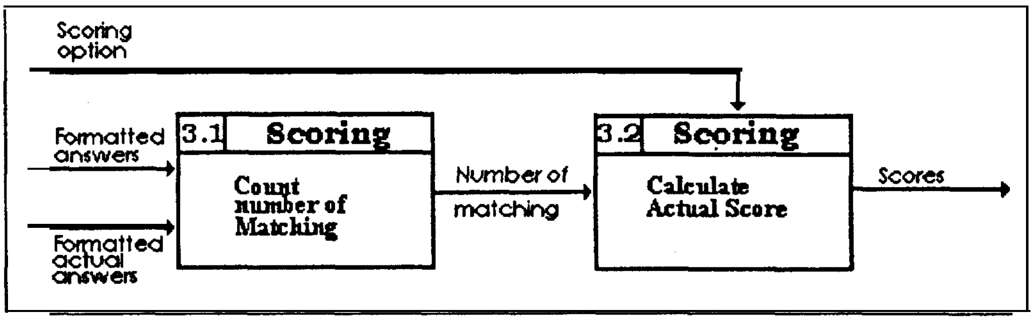


Figure 4.5 *Process 3*

Process 3 will do the scoring for the system. It takes 2 input, the formatted student answers and the formatted actual answers. Process 3.1 will compare the student's answers with the actual answers. It will count the total number of matches occurred in those comparisons, giving it as the output. Process 3.2 receives the output of process 3.1 as its own input, together with the scoring option. Depending on the scoring option, it will do the guessing correction process and calculate the actual score obtained by a student. The output of this process is the scores of all students.

Process 4

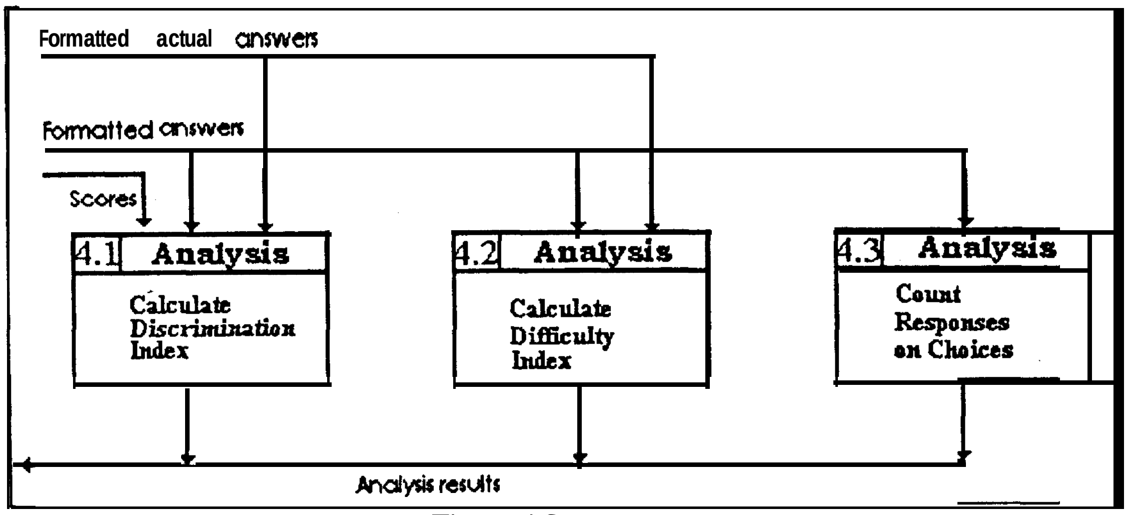


Figure 4.6 ***Process 4***

Process 4 contains 3 sub-processes. Process 4.1 receives 3 input and from those data, it will calculate the discrimination index of each item in the test. The output given is the discrimination index of all items in the test. In similar manner, process 4.2 only needs the student answers and the actual answers to produce the difficulty index. Process 4.3 will count the number of responses that choose each alternative in student answers. To do that, it only needs the student answers, and will give the result of the counting process as its output. So, the output of process 4 is the analysis results obtained from the 3 sub-processes.

Processes 5 and 6

Process 5 and process 6 are both primitive processes. They are the processes that do not have any sub-process. Both processes have simple tasks, reporting the input into more formatted forms. The data to be reported by processes 5 and 6 are 'scores' and the 'analysis results', respectively. The scores are then stored in the 'scores' data store and the analysis results are stored in 'analysis results' data store. External entity 'teacher' is given the access to these results.

In system analysis, usual extensions to DFDs are the use of data dictionary and the making of structured decisions. However, this project will not using data dictionary due to the small nature of this project. It is not needed since all data have been reduced to its lowest form in DFDs. Structured decisions process through structured English method is not discussed in this chapter. Instead, pseudocode, which is very close to structured English will be used in the implementation phase and will be shown in the next chapter. The next phase after system analysis is system design.

4.4 System Design

System design is the fourth phase in a system life cycle. In this phase, the information collected in earlier phases are used to design the system entity. The design process usually involves the system output, input, database files, user interfaces and data-entry. For this program, only program input, output, and user interfaces need to be designed. Database files' design is not needed since there will be no databases to be used and no relating files will be created. Data-entry design is not needed because data-entry will be done through other programs, before running this program.

4.4.1 Designing Program Output

This program will use 2 kinds of output; screen output and files output. The users of this program will be given the choices whether or not to write the results that they are seeing on the screen to an output file. If they want a hard copy of the results, they can write the results into a file and print them later. The output given will contain useful information normally gained after a test has been marked and analysed. The output of the marking part will include scores of each student, that is the number of correct answers and their percentage of right answer from all questions. The output of item analysis part will include the difficulty index, discrimination index and the count of each choice in a question.

4.4.1.1 Output File

The output file will be in ASCII text format, to make it easier to be imported into any word processor for printing purposes or into a spreadsheet for further calculation. For example, after using this program, a teacher might be interested in interpreting the test scores. This program will not be built for test scores interpreting, so it is important to make the scores easily accessible by other programs that will process and interpret the scores. To achieve that, the output file will be in the form of easily accessible ASCII text format. However, if one wants to quickly print the output, he/she can directly send the ASCII file to any line printer after quitting this program, without having to edit the file in a word processor first.

```
*********************************************************************
TEST INFORMATION
Class                 : XXXXXXXXXXXXXXXXXX
Test                  : XXXXXXXXXXXXXXXXXX
Number of students  :  99
Number of questions:  99

*********************************************************************
TEST   RESULTS
Names                    . Scores       Percentage
---------------------------------------------
XXXXXXXXXXXXXXXXXXXXX 99           99.99
XXXXXXXXXXXXXXXXXXXXX 99           99.99

*********************************************************************
ITEM   ANALYSIS   RESULTS
     Discr.        Diff.         Response   count
Q#  index          index         A      B      C      D      E      Omit
--------------------------------------------------------------------
9    9.99   .    9.99           99     99     99     99     99     99
9    9.99   :    9.99           99     99     99     99     99     99
9      9.99     it.99           99     99     99     99     99     99
```

Figure 4.7 *Output file layout*

Like illustrated in Figure 4.7, the output file is divided into 3 section: test information, test scores and item analysis results. The first section will contain the test name, the name of the class that has seat the test, the total number of candidate and the total number of questions in the test. The second section will show the result of test scoring, i.e. the test scores. Lastly, the third section will show the item analysis results. In both second and third section, the adjacent columns will be separated by a TAB, making it easier to import these columns into spreadsheet columns for further processing.

4.4.1.2 Screen Output

For screen output, this program will use a colour screen, which is able to display 16 colours text. While more advanced graphics screen adapter like VGA or Super VGA is commonly available nowadays, it is important to make sure that this program will be easily accessible by the educators that want to run it on a very basic PC. Furthermore, one of the target of this project is to provide a cheap alternative to machine-scoring and item analysis. So, this program will use only the text mode in displaying the output, because it will not require the more advanced monitors and more important, it is sufficient to serve the purpose of displaying the output.

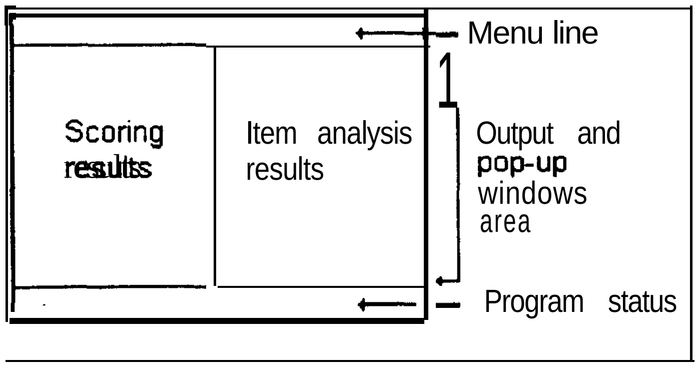


Figure 4.8 *The screen layout, showing the output area*

The screen is divided into 3 main sections, like depicted in Figure 4.8. The output will be shown in the middle area, at the centre of the screen. The output area is divided into 2 sections, one to display the scores and one for displaying the item analysis results. Each output area will be divided into columns of information. In both sections, if any area provided can not afford any surplus output, the user will be provided with a way to scroll up and down to see the excess output.

4.4.2 Designing System Input

After completing the DFDs, it can be clearly seen that the inputs to this program are the student answers and the teacher's answers, together with the teachers option in scoring the answers. Like the output, this program will use two forms of input. The input data will be taken from an ASCII text file, and the user will interact with this program using the keyboard. The keyboard will be used to take several program controls and options from the user. The input data, i.e. the answers of the test will be in ASCII text file format. This is the most simple file format that is supported by most word processors and can be prepared using the typical text editors such as Editor for DOS and Notepad for Windows.

The input data will not be typed directly by the user while using this program. Instead, the input data file must be ready before this program is entered. So, there is no need for designing a data entry form. What is needed is designing the input file format, so that it is easy to be prepared and can be easily read by this program. The input file format is shown in Figure 4.9.

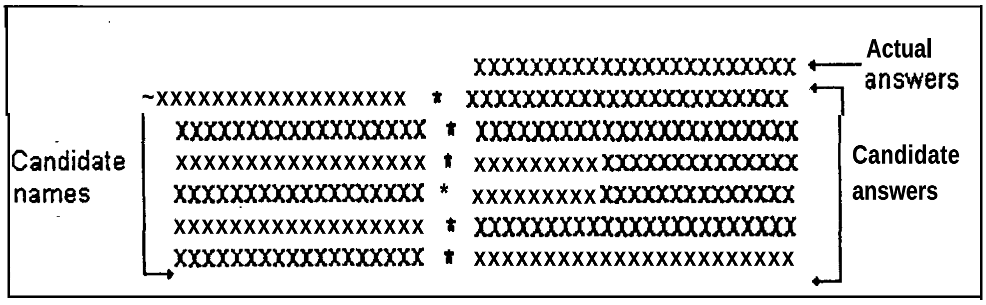


Figure 4.9 The ***inputfile format***

The first line of the file contains the teacher's answers, which will act as the actual answers to the test. The following lines after that contain candidate names and answers, with the name and answers be separated by an asterisk, '*'. The asterisk will act as an indication to the program that the next data is the candidate's answers, not his/her name. To simplify the task of typing in the input data file, the asterisk can appear exactly after the candidate's name and all the name length need not be the same, even though for user's own visibility, it will be permitted (see Appendix A for input data variations). Hence, all of the following will be a valid set of data:

```
John Smith * ABCCBBDEDDDAEBC
Michael Ritter * ABCCBBEEDDDAEBC
Jimmy Black    *   ABCCBBDEDEEXEBC
```

The name length will be restricted to only 20 characters. The total number of questions should not exceed 200 questions and in a test, this number should all be the same as the number of actual answers.

4.4.3 User Interface Design

4.4.3.1 Type of User Interface

The user of this program will choose the actions he/she wants to do by choosing the actions available from a set of pull-down menus. Pull-down menus were chosen to be implemented in this program because their uses are quite common in most programs currently available for PC. Furthermore, this can encourage inexperienced users to use this program, because the nature of choosing a selection from a pull-down menu is less error prone compared to numbered menu. In a pull-down menu, a user must highlight a choice and can read the text accompanying the highlighted choice before actually choosing it, making it less probable for the user to make a wrong choice.

4.4.3.2 Dialogues and Communications

Interactions between a user and this program will be done through the keyboard and the screen. The user will give all commands using the keyboard and this program will use the screen to response and give program output. While the use of mouse pointing device is currently popular, the consideration of time that have to be allocated for implementing mouse interfacing has retract the initial idea of using the mouse. For the means of consistency, the user will have to press ENTER each time he/she wants to choose any choice from the pull-down menu. The user will also use the same keys when doing the screen scrolling for browsing scores or item analysis results. This will speed the progress of the user in mastering this program

In taking the input such as the input and output file names, the user will be presented with a pop-up window prompting him/her for the filename. The length of the input will be restricted by the right edge of the window, ensuring that this program will get only right amount of characters from the user. This will also help to prevent the user from providing invalid filenames.

4.4.3.3 Feedback

In ensuring that the user will get the idea of what is the result of the action that has been taken, appropriate feedback will be given. Feedback will be given to:

1. Notify whether or not the input is in correct form.
2. Notify whether or not a request has been completed, and what should be done next.
3. Indicate that a selection is currently not available, and giving its reason.
4. Warn that the most current request will erase the current results.

Feedback will be given through pop-up windows, located at the centre of the screen.

This program will also have the current status indicator at the bottom of the screen, to tell the user the current state of the program. It will also show the keys that are available at those specific times. This is important in ensuring that the user always know what he/she is doing and what keys should be pressed. It will also help in indicating what should be the next reactions from the program.

In this system design phase, the program output, input and user interfaces were designed to meet the requirements. After the system design phase, the implementation phase is entered, where the coding of the design will be accomplished.

CHAPTER 5

THE IMPLEMENTATION OF THE ANALYSIS AND DESIGN

5.1 Introduction

The implementation phase of a system construction involves the software codes development by the programmers. The software development is based on the analysis and design results. In this program, pseudocodes are used as an intermediate step for developing the codes. The code development is done in three separate parts: the core program, the user interface and the integration of those two. The first and second parts were thoroughly tested to ensure they can reliably do their own jobs independently before integrating them. The program coding and compilation was done using Turbo C++ for DOS Version 1.1.

5.2 The Core Program

The core program is the inner part of the software that does the calculating job and produce the results, ready to be presented to the user by the user interface. This part is invisible to the program user. The functions in this part are activated only when they are requested by the interfacing part. The 3 main program parts that act as the core parts are data reading, scoring and item analysis.

5.2.1 Program Data Storage

The data that will be read from an input file will be stored in a 1 dimension (1-D) characters' array and 2 two dimensions (2-D) character's arrays (Figure 5.1). Actual answers will be stored in the 1-D array (Figure 5.1.a), starting with question number 1 be stored in array element 0. The maximum answers that it can store, i.e. the limit of the column, are determined by an integer constant, **MAXQUESTN.** The students' answers will be stored in -a 2-D array of characters(Figure 5.1.b), with the first student's answers be stored in the first row of the array, i.e. at-element 0,0. The students' names will be stored in another 2-D array of characters (Figure 5.1.c). The maximum name length that it can store is determined by the **MAXCNAME** constant. Both of these 2-D arrays have the same limit of total students that it can store, i.e. **MAXSTUDENTS.** When reading the input data, this array will be filled up until end of data is encountered or until all the rows have been filled.

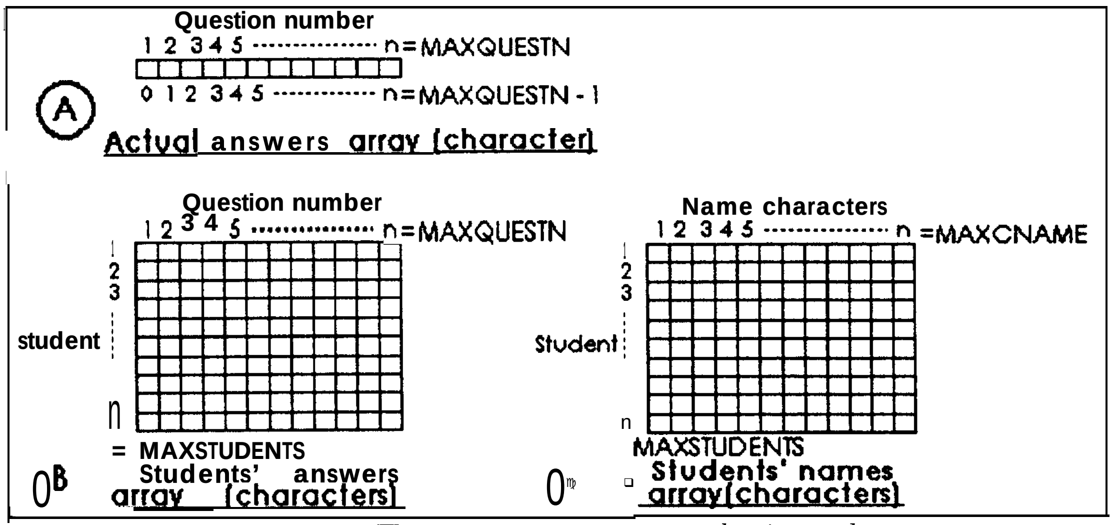


Figure 5.1 *The arrays that will store the input data*

The results of calculations will be stored in 4 arrays, like shown in Figure 5.2. The discrimination index will be stored in a float array (array A), starting from element 0 up to the maximum element, **MAXQUESTN** - 1. The difficulty index will also be stored in one float array (array B). Because array A and B have a common length, the discrimination and difficulty index arrays could also have been combined and stored in an array of structures that has 2 elements. However, this has not been implemented by considering the difficulties that will arise in sending them together to the user interface functions that will report the output to the user. The scores will be stored in an integer array (array C) and the response count results will be stored in a 2-D array of integers (array D).

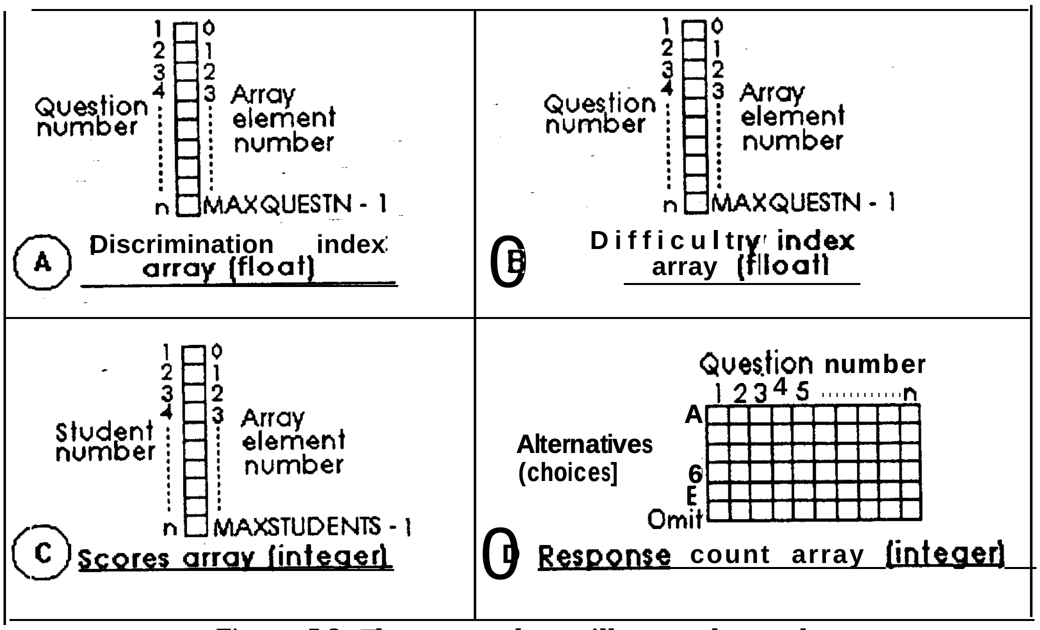


Figure 5.2 ***The arrays that will store the results***

5.2.2 Data Reading

Data to this program will be read from a text file. To simplify the user's task in preparing the file, it has been decided in the design phase that the teacher's and the candidates' answers are to be located in the same file. So, the reading part will take the answers from one file only. However, for programming flexibility, they will be done separately by different functions. The teacher's answers are read into the 1-D array shown earlier in Figure 5.1. The students' answers are read into the 2-D answers array. In doing this, the data reading part must ensure that the reading processes of both teacher's and candidate's answers are successful. If it fails to read in any answers, it will indicate it by returning a failure flag.

The pseudocode of teacher's answers reading:

```
OPEN input file
IF fail to open the input file
    RETURN failure flag
ELSE
    READ teacher's answers
    CLOSE input file
    RETURN success flag
```

The pseudocode of students' answers reading;

```
OPEN input file
IF fail to open the input file
    RETURN failure flag
ELSE
    READ teacher's answers
    number-of-questions = count from teacher's answers
    WHILE number of students not exceed maximum
        READ a student's name
        READ a student's answers into a 2D array
        IF total number of a student's answers <> number-of-questions
            Stop reading
            RETURN failure flag
        IF no more student's name exist
            BREAK, stop reading, exit from WHILE
    END WHILE
    CLOSE input file
    RETURN success flag
```

5.2.3 Scoring

Scoring involves 3 kind of data: students' answers, teacher's answers and the scoring option, i.e. whether or not to use guessing correction. The scoring function will first compare each student's answers with actual answers to get the raw score. The comparison is made easy by copying a row of the 2-D answers array (Figure 5.3). The resulting 1-D array will be sent into a counting function that will then return the number of correct answers. If the user had already opt to use guessing correction, the scores will then be corrected. All students' scores are stored in a 1-D integer array, shown earlier in Figure 5.2.c.

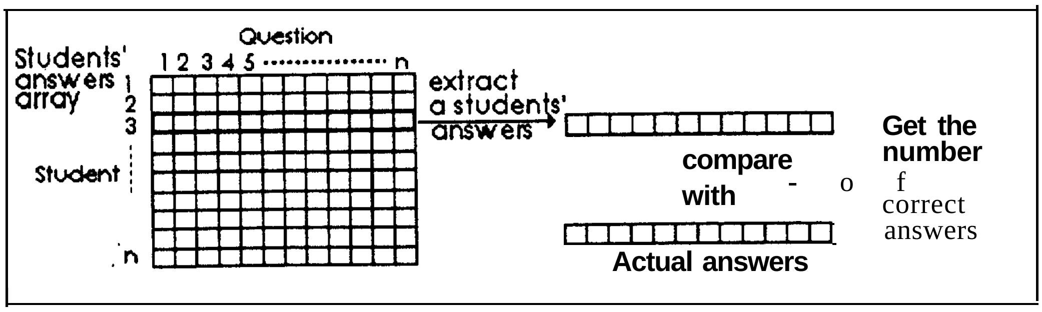


Figure 5.3 *Extracting a student's answers to do the scoring*

The scoring pseudocode:

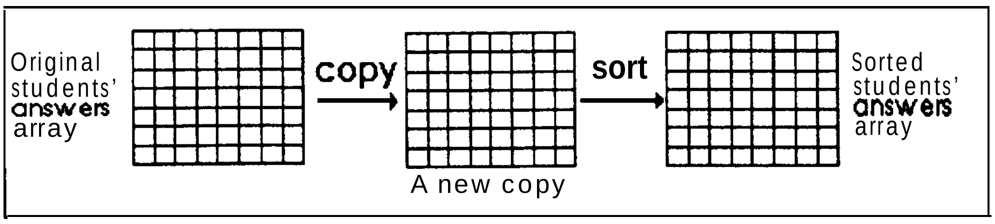
```
Declare a 1-D array, cur-student (to hold the answers for one student)
DO WHILE there are still student's answers
    copy a row into cur-student array
    count the number of correct answers for that student, comparing actual answers with cur-student
    IF want to use guessing correction
        count the number of wrong answers for that student
        update score with corrected score by doing guessing correction
    ENDIF
END DO
```

5.2.4 Item Analysis

Item analysis in this program consists of the calculation of discrimination index and difficulty index together with the counting of response on each alternative in each question. Each analysis is done separately by different functions.

5.2.4.1 Discrimination Index

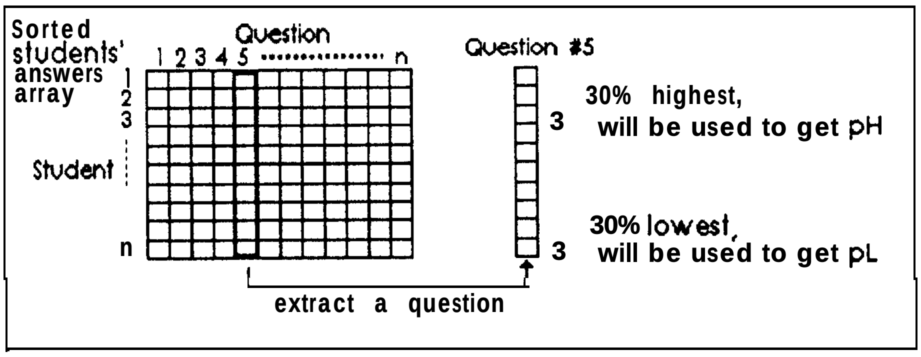
To calculate the item discrimination index, the students' answers must first be sorted according to the total marks obtained from the test as a whole. Since the original order of the students' answers array should remain intact, the function that will calculate the discrimination index must make a new copy of the array and then sort it before it can calculate the discrimination index. This is depicted-in Figure 5.4 below:



Original students' answers array
copy
sort
A new copy
Sorted students' answers array

Figure *5.4 Copying and sorting the students' answers 2-D array*

After sorting, the array element margins for 30% highest and 30% lowest students are then determined. In the loop that will calculate the discrimination index, the answers for each question are extracted from the answers 2-D array (Figure 5.5). The element margins are then used in getting the number of students that answered correctly in each part. The discrimination index is then calculated and stored in the 1-D float array shown earlier in Figure 5.2.a.



Sorted students' answers array
Question
1 2 3 4 5 .... n
1
2
3
Student
n
Question #5
3
30% highest, will be used to get pH
30% lowest, will be used to get pL
3
extract a question

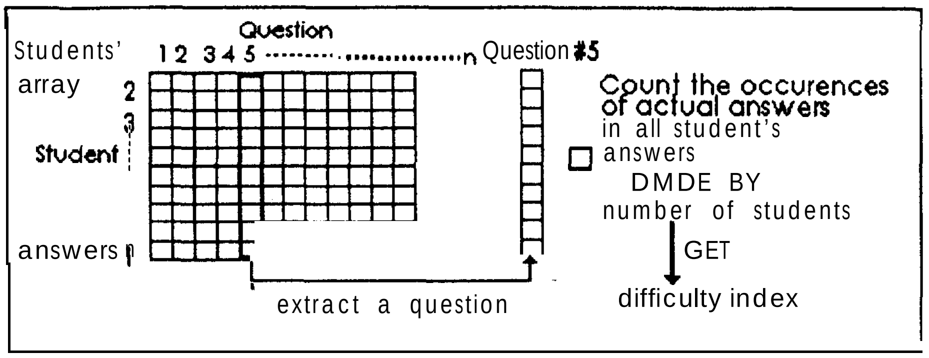
Figure *5.5* ***Extracting answers to calculate the discrimination index***

The pseudocode of discrimination index calculation:

```
copy answers 2-D array into new-answers
copy scores 1-D array into new-scores
sort new-answers and new-scores descendiigly
calculate 30% highest and 30% lowest array margins
DO WHILE there are still student's answers
    pH = number of correct answers in 30% highest / total students in the 30% part
    pL = number of correct answers in 30% lowest / total students in the 30% part
    disc-index = pH • pL
END DO
```

5.2.4.2 Difficulty Index

Calculating the difficulty index of a question involves counting the number of students who answered that particular question correctly. For simplicity and visibility, this program copy all the answers for that particular question before sending it to a function. This function will then count and return the number of occurrences of the actual answers in that particular question's array. This step is depicted in Figure 5.6.


Question
Students' array
1 2 3 4 5 ....n Question #5
2
3
Student
answers n
extract a question
Count the occurences of actual answers in all student's answers
DMDE BY number of students
GET
difficulty index
Figure *5.6 Extracting answers to calculate the difficulty index*

The pseudocode of difficulty index calculation;

```
Count total-no-students
Declare a 1-D array, cur-question (to hold the answers for one question)
DO WHILE there are still student's answers
    copy a column from 2-D array of answers
    count right-answers from that column alone
    diff_index = right_answers / total-no-students
END DO
```

5.2.4.3 Response Count

Doing the response count is much the same like counting the correct answers in calculating the difficulty index and scoring process. The same copy-and-pass procedure as already used in calculating discrimination and difficulty index is also used here. In response counting, the function that counts the number of occurrences of an alternative is called. The result of each counting is then stored in the 2-D array that stored the response count results shown earlier in Figure 5.2 (d).

The pseudocode of response count:

```
Declare a 1-D array, cur-question (to hold the answers for one question)
DO WHILE there are still student's answers
    copy a column from 2-D array of answers
    count responses for alternative 'A', then 'B', then 'C', then 'D', then 'E', then Omittance
END DO
```

5.2.5 Writing Output to File

In writing the results to the file, the printing function will use the file name supplied by the user interface part. If the file name is not valid, it will return an indication to the user interface that it can not open that file. If this function can open the file and write the results into the file successfully, it will then close the file and return a successful indication. In writing the results, it will select certain amount of information to be included in the output file, depending on the user options.

The pseudocode of output writing function;

```
Open output file
IF fail to open file
    RETURN failure flag
ELSE
    PRINT test information
    IF include-score
        PRINT names, scores, percentage scores
    ENDIF
    IF include-analysis-results
        PRINT analysis-results
    ENDIF
ENDIF
RETURN success flag
```

5.3 The User Interface

The user interface of this program depends much on the way the user will interact with the program. The user will spend most of his/her time doing one of three tasks: choosing an item from the pull-down menu, browsing the results and typing in several data such as input/output file names. So, the success of user interface of this program depends on the user interface of these three parts.

5.3.1 The Pull-down Menu Class

The pull-down menu in this program is used from the user-defined class specially coded for the use of pull-down menus. This class has been implemented to be a general purpose class that can be used in applications other than this project, too. The class has been carefully implemented so that its users (i.e. the programmers) can easily declare and use it. To declare a pull-down menu, this class needs 3 information: the menu top co-ordinate on the screen, the left co-ordinates and the strings that will appear in the menu. For example, the following code will declare a 'file' menu that contains 3 choices that will be activated at screen co-ordinate 10,10.

```
// step 1. declare the array string that will appear in the menu
char filetext[MAX_M_PANEL][MAX_M_CHARS] = {
'Load file', 'Close file', 'Quit',
};
// step 2. declare/construct a pull down menu named 'filemenu'
PullDown filemenu (10, 10, filetext);
// to activate:
int choice = filemenu.choose();
```

The constructor

Before a menu can be used, it must be constructed first, and this is done by a function called constructor. The constructor will first check the validity of the initialisation, and will also determine the screen co-ordinates of the menu.

<u>The pseudocode of the constructor:</u>

```
check the Xl and Y 1 co-ordinates -- not to exceed the upper/left screen border
calculate the longest panel suing, initialise X2
calculate the number of panels, initialise Y2
check the X2 and Y2 co-ordinates -- not to exceed the lower/right screen border
get the menu-strings for each panel
fill in the empty right part of each string with spaces
```

To choose an item

Choosing an item from a pull-down menu involve navigating through its panel. This is done by using the arrow keys, UP and DOWN. Once we arrived at the panel we want to select, we will press the ENTER key. This class implemented the pull-down menu panel selection using this conventional method. If a programmer wants to activate a pull-down menu he/she has declare earlier, it is done by calling the choose() function, and assigning its return value into an integer variable (see the sample code fragment above). The choose() function will return the panel number the user has selected. If, however, the user does not choose any panel from the menu (i.e. by pressing ESC or left arrow or right arrow), this function will return a number equivalent to those keys. This returned number can then be used by the programmer in determining an appropriate action. One example of its usage is if a user has pressed the right arrow, another pull-down menu that sits at the right of the current menu can then be activated.

The pseudocode of choose() function:

```
store the current menu area into buffer
show pull-down menu, highlight first panel
wait for keystroke
DO WHILE true
    IF keystroke = down-arrow
        highlight the panel below current panel
        IF current panel is the bottom panel
            highlight the top panel
        ENDIF
    ELSEIF keystroke = up-arrow
        highlight the panel at the top of current panel
        IF current panel is the top panel
            highlight the bottom panel
        ENDIF
    ELSEIF keystroke = ENTER
        restore screen buffer
        return the number of current panel
    ELSEIF keystroke = LEFT / RIGHT / ESC
        restore screen buffer
        return the number of that keystroke
END DO
```

5.3.2 Browsing the Results

For displaying output, the screen is divided into two parts. The scores will be shown on the left of the screen and the item analysis results will be shown on the right of the screen. Since the total number of students and the total number of questions are usually greater than the screen rows, the screen can not afford to display either results in one screen. So, a scrolling method must be developed to browse through both scoring and analysis results.

All results of this program to be displayed were stored in arrays (see Figure 5.2). The array element number can be used as a reference to emulate the position of screen margins to be displayed to the user. This is visualised in Figure 5.7. The top margin will first hold element 0, and will then be increased when the user pressed DOWNARROW. If the user pressed PAGEDOWN, the margins areincrementedas many as a screentall. The processof margins changing will continue, like depicted in the pseudocode for the result browsing, and will only stop when the user has pressed ESC to stop browsing.

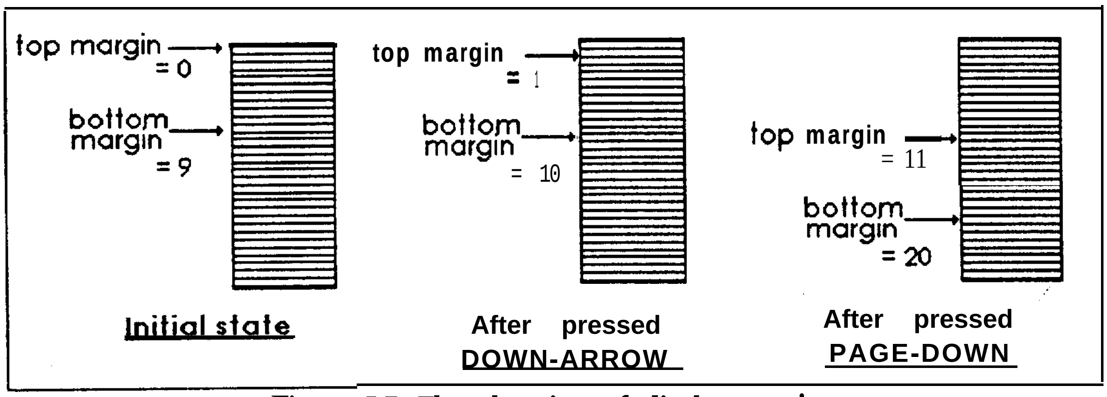


Figure *5.7 The changing of display margins*

The pseudocode of result browsing;

```
initialise top and bottom margin (top = 0, bottom = screen_tall)
DO WHILE true
    SELECT CASE keypressed
        CASE PAGE_DOWN
            increase top and bottom margins by one screen
        CASE PAGE-UP
            decrease top and bottom margins by one screen
        CASE ARROW_DOWN
            inaease top and bottom margins by 1
        CASE ARROW_UP
            decrease top and bottom margins by 1
        CASE CTRL_HOME
            set margins like early state (top = 0)
        CASE CTRL_END
```

```
            set bottom margin = max_array_element (go to far end)
        CASE ESC
            ❒♏♦♦❒■
    END SELECT
END DO
```

5.3.3 Prompting for A String

Although this program will read most of its input data from an input file, it still needs some string input from the user when the program is running. These strings have to be typed in by the user. These include the input data file name, the output file name, and the information of the test being analysed. A special function has been created for the purpose of taking string input from the user while the program is running. This function will receive the pointer of the suing to be input, and will modify its content, depending on the keys pressed by the user. The user can only exit from giving this input by pressing ENTER or ESC. If ESC is pressed, the string will be set to contain nothing.

<u>The pseudocode of string prompting:</u>

```
put cursor at starting position
Do
    get keypressed
    IF keypressed = ENTER
        done = true
    ELSEIF keypressed = BACKSPACE
        retract cursor one position
    ELSEIF keypressed = ESC
        set string contains nothing
        done = true
    ELSE
        accumulate keypressed
        advance cursor one position
WHILE not (done)
```

5.3.4 Managing Pop-up Windows

There are many occasions in this program that a window has to be popped-up to show messages to the user. Windows are also popped-up before the string prompting function is called. The pop-up window pseudocode is shown below. The pop-up window used in string prompting is much the same as this pseudocode, except in 2 places. First, it has to include the length of the string to be input in calculating the co-ordinates and the buffer size. Secondly, it has to wait until the user finished giving input before the screen area can be restored.

Message pop-up windows general pseudocode:

```
calculate width using string message to be shown
calculate left X co-ordinate and right X co-ordinate
calculate screen area size
capture screen area into screen buffer
wait for keypressed
restore screen area
```

5.4 The Integration of Core Program and User Interface

The core program and the user interface must be integrated by an outer layer. This most outer layer is where a user of this program can navigate in the main pull-down menus. Appropriate core action will be taken according to the selection of the pull-down menus. In the program code, this outer layer is located in the main program. The pull-down menus are declared and activated from this part. When entering this program, the user is first presented with this menu of pull-down menus. To navigate between these menus, the user can use either RIGHT or LEFT arrow keys. If a user wants to select a choice, he/she must press ENTER when the choice is being highlighted. The logic of this outer layer user interface is explained below.

The general pseudocode of navigating through the main menu:

```
activate first pull-down menu
Do
    wait for keypressed
    IF keypressed = RIGHT
        activate the pull-down menu at the right
    ELSEIF keypressed = LEFT
        activate the pull-down menu at the left
WHILE true
```

'Activate' in the-above pseudocode means highlighting the menu title, show the pull-down menu and ready to take choices from this menu. This general idea has contribute to a more specific idea of using arrays of menus to represent pull-down menus. The array element number will act as a reference number, to be used when a menu is to be activated or de-activated.

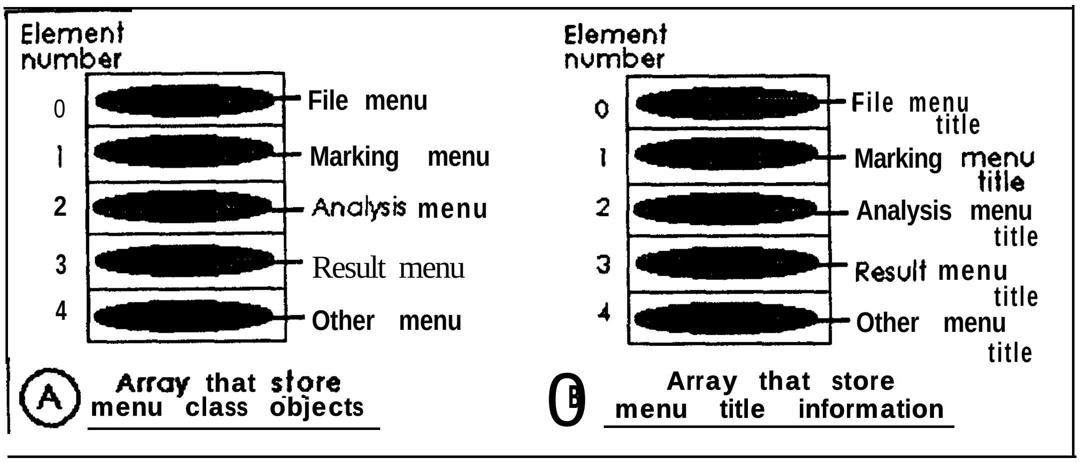


Figure *5.8 Arrays that store the menus*

Two arrays will be used to store the menus, like shown in Figure 5.8. The first one (array A) will store the actual menu, and the second one (array B) will store the information about the menu title. They will be referred in those 2 arrays with the same element numbers. When a pull-down menu is to be activated, the title information array (Figure 5.8(b)) will he used to highlight the current title and dimmed the previous title. The actual pull-down menu is then activated by referencing the pointer of its element in the array in Figure 5.8(a). By changing this element number, the menus will then be activated one by one, depending on the keys that have been pressed by the user.

The detail pseudocode for navigating in main menu;

```
declare an array of structures, storing menu title information
declare menu strings
declare pull-down menu objects
declare an array, storing all the above pull-down menus objects
let current-menu-element = 0
Do
    take pointer to the current-menu-element
    let choice = activate current-menu-element
    IF choice = RIGHTARROW
            increment current-menu-element
    ELSEIF choice = LEFTARROW
            decrement current-menu-element
    ELSE
        take appropriate action, using SELECT CASE . . .
WHILE true
```

5.5 Conclusion

The coding phase in this program has been developed in three different parts: the core program, user interface and the integrating part. The core program is concerned about getting the results from input data. The user interface is responsible in getting the user response and acts as a platform where the user can interact with the core program. The integrating part is responsible in preparing a way of giving choices to the user in selecting an appropriate action while running this program.

CHAPTER 6

USER DOCUMENTATION

6.1 Requirements

You must run this program on a PC running with at least 256kb RAM, operating on DOS operating system version 4 or higher. The PC must also have a colour monitor and a keyboard attached to it. The colour monitor must be able to display text in 16 colours. You would not need a mouse to use this program. You need not to have a hard disk, because you can also run this program from a floppy disk.

6.2 Starting and Quitting The Program

To start this program, change your current directory to the directory that contains this program executable file, named **MCQTEST. EXE.** If the program resides in your floppy disk, insert your disk into the disk drive first and change your current drive into your floppy drive. Type the program name once you are ready. For example, if you are running it from your floppy drive **A: ,** type **MCQTEST:**

```
A:\>  MCQTEST
```

This program will then start and you will be given the program screen.

You can quit from using this program by highlighting the Quit panel from File menu, and press **ENTER. YOU** will be brought back to the DOS prompt.

```
A:\>
```

6.2.1 Screen Layout

You can make yourself familiar with the screen (Figure 6.1) before proceeding using this program. You will choose your actions from the pull-down menus located at the top of the screen. You can move between these menus with LEFT and RIGHT arrow keys and you can select a choice by pressing **ENTER**. The 2 lines at the bottom of the screen will give you 2 useful information. The first line will show the current program status, i.e. what are you doing at the time. The second line will show the keys on the keyboard that you can press at that specific time. The results of your actions will be shown in the large centre area of the screen. The left section will be used to display the scores obtained by the students in a test. The right section will be used to display the item analysis results, i.e. the discrimination index, difficulty index ad the ***response*** counts of the test items. You will be using this program by navigating in just 3 areas: the pull-down menus, the score browsing section and item analysis results browsing section.

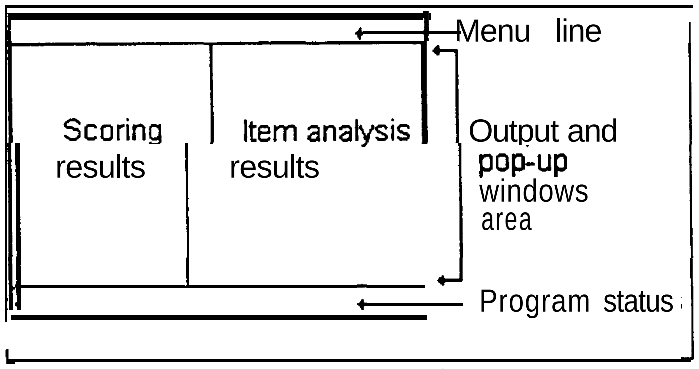


Figure 6.1 ***Screen layout of the program***

6.2.2 Main Menu

This program has 5 pull-down menus to choose from (Figure 6.2).

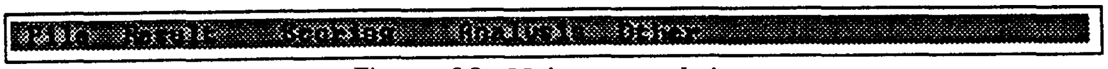


Figure 6.2 ***Main menu choices***

After entering this program, the File menu will be activated for you. You can move between all these menus with LEFTARROW and **RIGHTARROW** keys. Forexample,ifyou want to move to the Result menu from File menu, you can press **RIGHTARROW** key. However, if you just entered this program, you can only choose from the File menu. Any attempt to choose from other menus will pop-up an error message. You are not allowed to completely leave this main menu. Although you can do tasks after selecting from this menu, after you completed your task, you will be brought back to this menu.

6.3 The Menu Items

6.3.1 File Menu

File menu has two choices (Figure 6.3).

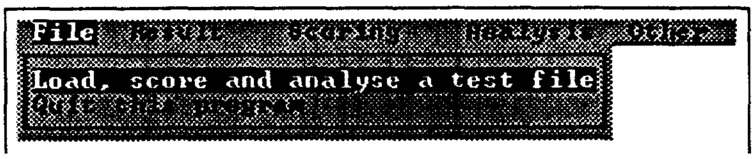


Figure 6.3 ***File menu***

The first choice is to load, score and **analyse** a test file. If you press ENTER on this panel, and there are no currently unsaved tests result, you will be presented with a pop-up window asking you to type in the name of the data file (see section 6.4 for details on typing file name and section 6.5 for input file details). The data from that file will be read, the answers will be automatically scored and the items will be analysed. A pop-up window suggesting you to use the Result menu to browse the results will appear.

The second option from this menu is used for quitting from the program. If you have load a data but the results have not been saved, a warning message telling you that the results have not been saved will appear. You can choose to cancel quitting, and return to the main menu. This program will quit and will not giving this message if you have saved the results to an output file, or-you have not load a file at all.

6.3.2 Results Menu

The Results menu is used for browsing and saving the results of scoring and item analysis (Figure 6.4). The first two choices are used to browse the scores and the item analysis results on the screen. The third panel is used to save the results to an output file. If you choose this third choice, you will be given another pull-down menu prompting you to choose the results that you like to include in the output file (see section 6.7 for details of output file).

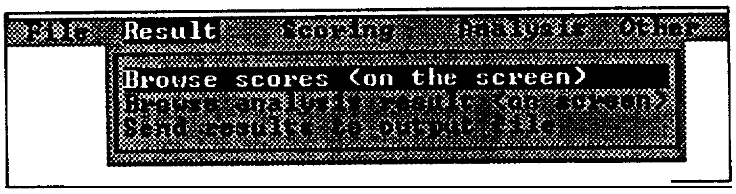

File
Result
Scoring
Analysis
Other
Browse scores (on the screen)
Browse analysis results (on screen)
Send results to output file

Figure 6.4 *Results menu*

6.3.3 Scoring Menu

The Scoring menu will give you the options to change the scoring methods, like shown here on Figure 6.5.

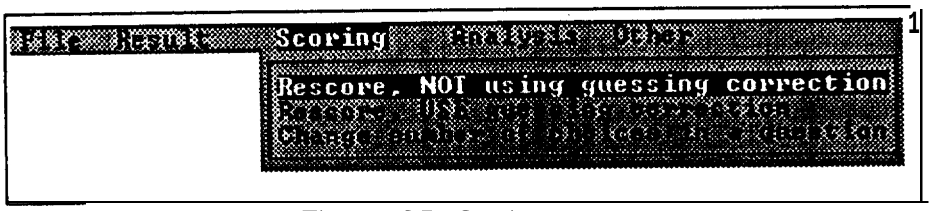

File
Result
Scoring
Analysis
Other
Rescore, NOT using guessing correction
Rescore, USING guessing correction

Figure 6.5 *Scoring menu*

Selecting the first option will make the answers to be re-scored, giving the scores in raw marks. Choosing the second option will make the students' answers be re-scored, and the raw marks are corrected before being presented to you. The third choice is to be used when you want to use guessing correction with the number of alternatives in the test questions is less than 5. For example, to use guessing correction on true/false test items, you can choose this third choice, then choose “2 choices” from its submenu, and then re-score using guessing correction (choose the second menu choice). By default, the guessing correction will calculate the corrected scores by assuming that there were 5 alternatives (A, B, C, D, E) in the test questions to be analysed. Each time you select to re-score, you will be automatically entered into the score browsing mode, enabling you to look at all the scores. You will have to press **ESC** to return to main menu.

6.3.4 Analysis Menu

The fourth selection from the main menu is Analysis menu (Figure 6.6). This menu presents you with an opportunity to display the response counts in the form of percentage of students that choose the alternatives, instead of just the raw numbers. If you select one of the options from this menu, you will enter the mode where you can browse the analysis results, with the response counts format shown according to your preference.

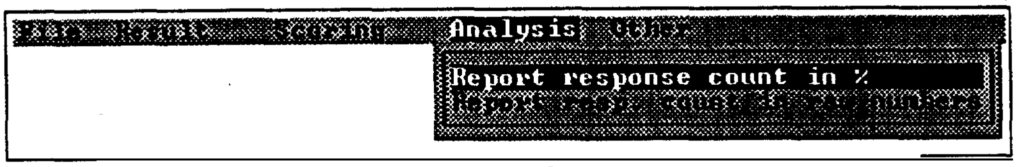


Figure 6.6 *Analysis menu*

6.3.5 Other Menu

The last menu’is the Other menu, which has 3 selections (Figure 6.7).

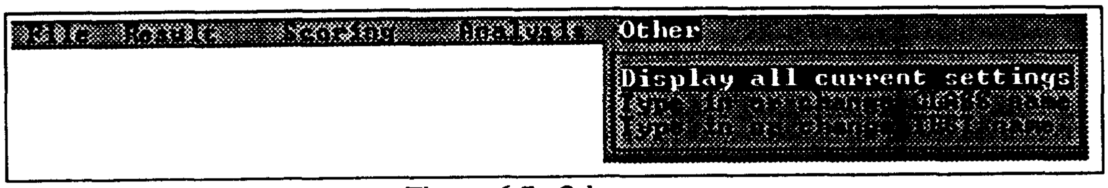


Figure 6.7 *Other menu*

The first selection will display all the current settings and the test information. The current settings shown include whether or not the guessing correction has been used and the number of alternatives in a question. The test information includes the class name, the test name, the total number of students and the number of questions. The second choice will let you type in the name of the students' class whose answers are being dealt with. The third choice will let you type in the name of the test itself, i.e. the name of the subject or the module. These names will be used in printing the test information into the output file and in displaying the current settings.

6.4 Typing In the File, Test or Class name

When you choose to open a data file, or to write the results to an output file, you will be prompted for the file name. You will have to type in the file name in a pop-up window, like Figure 6.8.

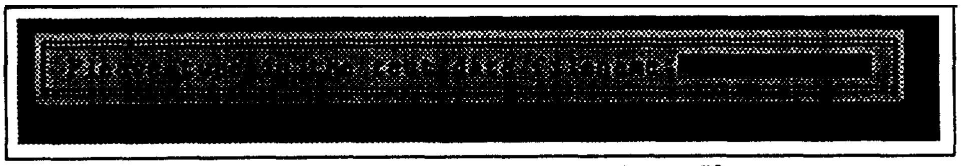


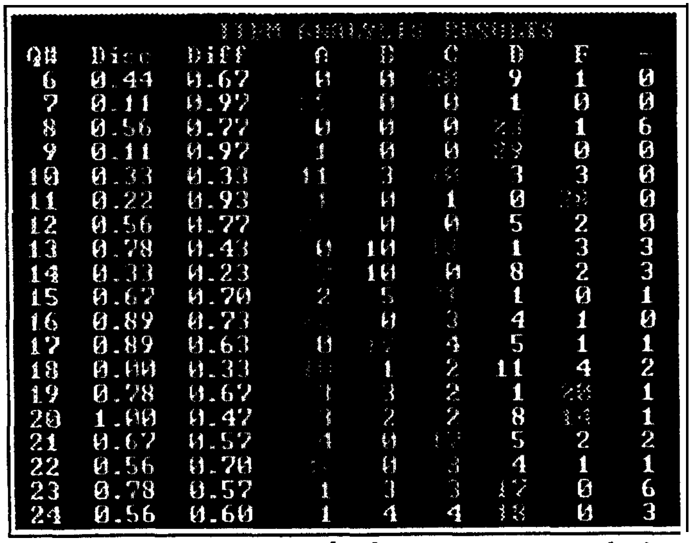
Figure 6.8 *Prompt for typing in the input file name*

When you are prompted with the file name, you must type in a valid ASCII text file that contains the test answers. Like indicated at the status line at this time, you can type in any character that is valid in a file name. If you have type in a few characters, but realise that you have made a mistake, you **can** correct the mistake using the **BACKSPACE** key. Your input length at this prompt is restricted by the right edge of the pop-up window. You can complete your task of filling in the file name by pressing the ENTER key.

When you-want. to input the class name-whose answers are being dealt with and the test subject name, you will -be given another prompt, which also looks like Figure 6.8. The difference is the sentence that prompts you for the names and the length of **input, which is** longer..

6.5 Browsing the Results

After you have load in the input data, you should browse the results of scoring and item analysis by selecting Result menu. When you are browsing the scores, you can scroll through the *scores* by pressing Up Arrow, Down Arrow, Page Down, Page Up, Control Home and Control End. Up Arrow and Down Arrow will let you scroll one line at a time, Page Up and Page Down will let you scroll one page at a time, while Control Home and Control End will take you to the start and the end of the **list.** Figure 6.9 shows an example of the results from the item analysis. For response count, it shows the number of students that opt for the correct answer in a quite different colour from other alternatives.



```
                      ITEM ANALYSIS RESULTS
Q#   Disc   Diff     A    B    C    D    F    -
 6   0.44   0.67     0    0   20    9    1    0
 7   0.11   0.97    29    0    0    1    0    0
 8   0.56   0.77     0    0    0   23    1    6
 9   0.11   0.97     1    0    0   29    0    0
10   0.33   0.33    11    3   10    3    3    0
11   0.22   0.93     1    0    1    0   28    0
12   0.56   0.77    23    0    0    5    2    0
13   0.78   0.43     0   10   13    1    3    3
14   0.33   0.23     7   10    0    8    2    3
15   0.67   0.70     2    5   21    1    0    1
16   0.89   0.73    22    0    3    4    1    0
17   0.89   0.63     0   19    4    5    1    1
18   0.00   0.33    10    1    2   11    4    2
19   0.78   0.67     1    3    2    1   20    1
20   1.00   0.47     1    2    2    8   14    1
21   0.67   0.57     4    0   17    5    2    2
22   0.56   0.70    21    0    3    4    1    1
23   0.78   0.57     1    3    3   17    0    6
24   0.56   0.60     1    4    4   18    0    3
```

Figure 6.9 ***Sample results from the item analysis.***

6.6 Input File

The test actual answers and students' answers should be ready before you start this program, since this program did not provide you with a facility to input them. The file can be prepared using any text editor or word processor, as long as it is saved as the ASCII text file. You can see the input file samples provided with this program (see Appendix A). The general format of the file is shown in Figure 6.10.

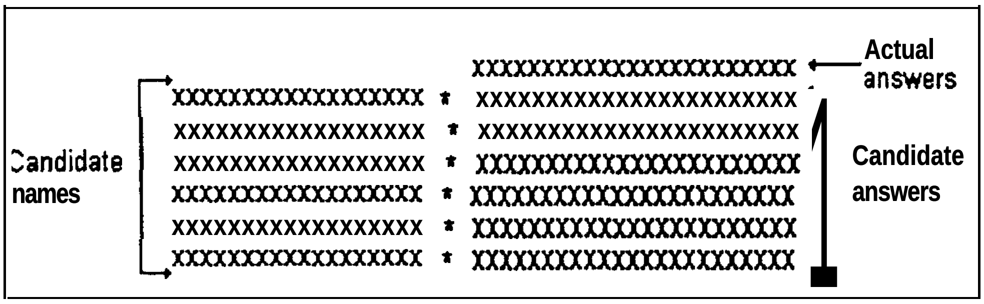


Figure 6.10 *Input file format*

The test actual answers must appear in the first line of the file, with no spacing between each answer. The total of answers should not exceed 150, since all answers after the 150th answer will be ignored by this program. For your convenience, you can adjust the position of the actual answers so that they are in the same column as the students' answers, like shown in Figure 6.10, as long as the are preceded only by spaces. The next lines should contain the students' names and their answers. The name length should not exceed 20 letters, including spaces. If this length is exceeded, the exceeding letters will be ignored. After a name, there should be an asterisk. If a student's name has less than 20 characters, you can put the asterisk exactly after the name, even though you may prefer to make it parallel with other names. The student's answers should appear after that asterisk. The total number of a student's answers should be the same as the teachers' answers. If one of the students has a different number of answers, this program will present you with a message telling you this error, and the data file will not be loaded. You can exit this program and edit the data file before entering it again.

6.7 Output File

An output file is created when you save the results. The format of the file will depend on the way you saved the file. When you choose to save it, you will be given a pull-down menu asking you for the information that you want to include in the output file (Figure 6.11).

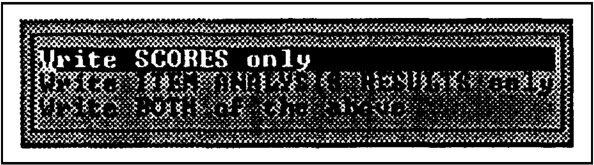


Figure *6.11 Choosing the outputjile contents*

So, the output file format will depend on your choice from this menu. One information that will be written whatever selection you made is the test information. This includes the test name, the class name, the total students and the total questions. Test scores and item analysis results will only be written if you choose to write them or you choose to include both of them in the file. In this case, the file format will be like shown in Figure 6.12.

```
***************************************************************************
  TEST INFORMATION
  Class                 : XXXXXXXXXXXXXXXXXXX
  Test                  : XXXXXXXXXXXXXXXXXXX
  Number of students :  99
  Number of questions:  99
  *******************************************************************t**
  TEST RESULTS
  Names                      Scores        Percentage
  ---------------------------------------------------
  XXXXXXXXXXXXXXXXXXXXXXX 99            99.99
  XXXXXXXXXXXXXXXXXXXXXXX 99            99.99
  *********************************************************************
  ITEM ANALYSIS RESULTS
      Discr.         Diff.          Response   count
  Q#  index          index          A      B      C      D      E      Omit
  --------------------------------------------------------------------
  9      9.99      9.99           99     99     99     99     99     99
  9      9.99      9.99           99     99     99     99     99     99
```

Figure 6.12 *Output file layout*

This program is not intended to do the score processing. It does not even give the simplest information-about the scores, for instance the average of the scores. The scoring done **is** just for providing the scores to the teachers and supplying input to enable the item analysis. So, after getting the results, you may need to make further processing on the scores, like scaling, or you may want to make some statistical analysis that is not supported by this program. To do that, you can easily import this output file into other programs like a spreadsheet, because the adjacent columns are separated by a **TAB**. Most spreadsheets are able to import data from ASCII text file and the columns can easily be imported into the cells if they are separated by the **TAB**. So, even if this program has not supported the more advance score processing, it prepares a method to link the way of doing it.

CHAPTER 7

CONCLUSIONS

7.1 Project Objectives Achievement

The main goal of this project is to built a computer program that can act as a tool to score and analyse multiple choice question tests. The program should also small, cheap and easily accessible by all educators, i.e. they can use it without having to face so many problems. The program has been completed and is ready to be used. Although it can cope with what has been a cumbersome problem for educators, it is small enough that it can reside in a diskette, together with the data files. It can be run on a typical PC that can be found in most schools at present time. It can do its job in a PC with just 256kb of conventional RAM and does not require a hard disk. The program operations are based on the actions taken while the user is using the pull-down menu. So, the user does not have to know complicated commands to use it. This program also gives an adequate amount of message when an error occurred, and indicates what should be done when an error occurred. So, it can be said that this program is not difficult to use by an average user. The fact that it is small, does not require advance skills on operating it, and can run on a PC with low specifications contribute much in achieving the target of this project. So, as a conclusion, this project has successfully achieved its goal.

7.2 Further Enhancements

While the goal of this project has been achieved, there are still further enhancements can be made to rectify the program. Some re-engineering may be need'to be done to design a better program, but that will not make it hard to improve it, because the program functions have been coded so that they are as objective as possible. The core program that does the calculation jobs was developed separately from the user interface part that will take user responses. So, if more output is required, the core program can be modified; If the interface is later found out to be flawed somewhere, it can be re-coded and tested separately. Furthermore, there is a pull-down menu class that is ready to be used anytime. So, any addition in program selection can be accomplished without taking so much time and effort

Possible extension to this project would be to incorporate it with an automatic answer reading procedure. Currently, the computer program developed through this project requires that the test answers be typed in into a text file before being processed. May be this fact will discourage some educators from using it if they think that the task of preparing the data file is also a cumbersome job. One possible solution is to incorporate this program to process the answers that are automatically entered through an automated testing, using computer terminals as test taking stations. Other possible solution is to have the students mark their answers to the test questions using specially designed answer forms. The forms can then be scanned using a scanner to trace the answers.

References

1. Barclay, Kenneth A., Gordon, Brian J., ***C++ Problem Solving and Programming;*** Hertfordshire, Prentice Hall International, 1994.

2. Gronlund, Norman E., ***Measurement and Evaluation in Teaching. 4th*** Ed. New York, Macmillan Publishing, 198 1.

3. Kendall, Kenneth E., Kendall, Julie E., ***systems Analysis and Design.*** 3rd. Ed. Englewood Cliffs, Simon And Schuster, 1995.

4. Lyman, Howard B., ***Test Scores and What They Mean.*** 3rd Ed. Englewood Cliffs, Prentice-Hall, 1978.

5. Peddie, Bill, White, Graham, ***Testing In Practice.*** Auckland, Heinemann Educational Books, 1978.

6. Ray, John R., Davis, Lloyd D., ***Computers in Educational Administration.*** California, Mitchell Mcgraw-Hill, 199 1.

7. Schildt, Herbert, ***C+ + The Complete Reference.*** Calfornia, Osborne Mcgraw-Hill, 1995.

8. Schildt, Herbert, ***Using Turbo C++.*** Calfornia, Osborne Mcgraw-Hill, 1990.

9. Sumner, Ray, ***The Role Of Assessment in Schools.*** Windsor, Nfer-Nelson, 199 1.

10. Wiersma, William, Jurs, Stephen ***G., Educational Measurement And Testing.*** 2nd.Ed. Massachusetts, Simon And Schuster, 1990.

11. Wood, Robert, ***Assessment and Testing, A Survey** Of **Research.*** Cambridge, Press Syndicate, 1991.

APPENDIX A
DIFFERENT VALID FORMATS OF INPUT FILE

All of the followings are examples of valid answers order in the input file for this program.

1. All answers-in the same columns, with asterisks on the same columns in each line.

```
                         ABAEECADDCEACACEEECECADBBD
Hillary Best           • ABAEEDA-DCEDBDBEECECADCAD
Alexandra Smith        * ABAEDCADDBEABACCACECEECAD
Lindsay Scott          * ABDEECADDCEACBCBECECADBBD
Mike Johnson           * BBEECADDCBACBCEECDECADCAD
Johaan Young           * DD--CCADDBEDBDBDECECEECAD
Mark Brown             * ABAEEDADDCEACACDEDECADCA-
Patrick   McNamara     *    ABDEECADDCELACBCAECECADCAD
```

2. All answers in one same column, except actual answers.

```
ABABECADDCEACACEEECECADBBD
Xillary Best           * ABAEEDA-DCEDBDBEECECADCAD
Alexandra Smith        •  ABAEDCADDBBABACCACECEECAD
Lindsay Scott          • ABDEECADDCBACBCBECECADBBD
Mike Johnson           • BBEECADDCBACBCEECDECADCAD
Johaan Young           * DD--CCADDBEDBDBDECECEECAD
Mark Brown             * ABAEEDADDCEACACDEDECADCA-
Patrick McNamara       • ABDEECADDCEACBCAECECADCAD
```

3. All answers in one same column, with asterisks not on the same column in each line,

```
                  ABABECADDCEACACEEECECADBBD
Xillary Best * ABABEDA-DCEDBDBEECECADCAD
Alexandra Smith • ABABDCADDBBABACCACECEECAD
Lindsay Scott * ABDEECADDCEACBCBECECADBBD
Mike Johnson * BBEECADDCEACBCEECDECADCAD
Johaan ✡□◆■♑ • DD--CCADDBEDBDBDECECEECAD
Mark Brown • ABAEEDADDCEACACDEDECADCA-
Patrick McNamara * ABDEECADDCEACBCAECECADCAD
```

4. Answers in different columns, with asterisks not on the same column in each line, and actual answers starting at the first column.

```
ABAEECADDCEACACEEECECADBBD
Xillary Best • ABAEEDA-DCEDBDBEECECADCAD
Alexandra Smith  * ABAEDCADDBEABACCACECEECAD
Lindsay Scott * ABDEECADDCBACBCBECECADBBD
Mike Johnson       • BBEECADDCBACBCEECDECADCAD
Johaan Young     *     DD--CCADDBEDBDBDECECEECAD
Mark Brown        • ABAEEDADDCEACACDEDECADCA-
Patrick McNamara • ABDEECADDCEACBCAECECADCAD
```

APPENDIX B
THE CODE OF PULL-DOWN MENU DEFINITION FILE

```
/////////////////////////////////////////////////////////////////
// General purpose pull down menu definition file              //
//          Done          on          Sat          22/7        //
//        By:        Mohamad        Zamberi        Saad        //
//     For the completion of MSc of Information Technology //
//  Completed   between   Jun   1995   -   September   1995    //
/////////////////////////////////////////////////////////////////

#ifndef PULL_DOWN_H
#define   PULL-DOWN-H

// keys on keyboard that will be used when choosing an item
const int UP-ARROW = 72;
const int DOWN-ARROW = 80;
const int LEFT-ARROW = 75;
const int RIGHT-ARROW = 77;
const int ENTER = 13;
const int ESC = 27;

#include <conio.h>
// menu color
const  int  N-TEXT-COLOR  =  BLUE;   // blue on white
const int N_BG_COLOR = WHITE;
const int H-TEXT-COLOR = WHITE; // white on blue
const int H_BG_COLOR = BLUE;


const int MAX_M_CHARS = 40; // maximum chars in a menu panel
const  int  MAX-M-PANEL  =  4; // maximum panel in a pull down menu

class PullDown {

  private:
    int Xl, Y1, X2, Y2;
    char  menu_str[MAX_M_PANEL]  [MAX-M-CHARS];

  public:
    // constructor
    PullDown(int x_1, int y-1,  char str_of_menu[MAX_M_PANEL][MAX_M_CHARS]);

    // choose an item from the currently shown pull down menu
    int choose ();

    // display all the choices in every panel
    void init_show ();

    // change the hlite panel from prev panel to cur panel
    void  change-hlite  (int  prev,  int  cur);

}; // end class

#endif
```

APPENDIX C
THE CODE OF PULL-DOWN MENU IMPLEMENTATION FILE

```
////////////////////////////////////////////////////////////
//   Implementation   of   pull-down      menu   class   //
//          Done:          Sat.          22/7/95          //
//       By:          Mohamad        Zamberi        Saad    //
//     For the completion of MSc of Information Technology //
//  Completed   between   Jun   1995   -  September  1995  //
////////////////////////////////////////////////////////////

// example use: to declare a 'file' menu that contains 3 choices
//    that will be activated at co-ord 10,10
//  // step 1. declare the array string
// char helptext[MAX_M_PANEL] [MAX-M-CHARS] = {
//    "Load file', 'Close file', 'Quit",
//  };
//  // step 2. declare/construct the pull down menu
//  PullDown filemenu (10, 10, filetext);
//
// to activate: int choice = filemenu.choose();

#include <iostream.h> // needed by cerr
#include <stdlib.h>   // needed by exit()
#include <string.h>   // needed by strlen() and strcat()
#include "pulldown.h"

// constructor
PullDown::PullDown (int x_1, int y_1,
        char str_of_menu[MAX_M_PANEL][MAX_M_CHARS]) {

   // 1. check the length -- not to exceed the upper/left screen border
   if (x_1 < 3 II y_1 < 3) {
     cerr << "\nCan not init. menu: border (upper/left) out of screen.\n";
     exit (-1);
   }

   // 2. calculate the longest panel string, to get X2
   int lgst-str = 0;
   for (int i = 0; i < MAX-M-PANEL; i++) // note : start from 0, to max-1
     if (lgst-str < strlen(str_of_menu[i]))
       lgst-str = strlen(str_of_menu[i]);
   x2 = x_1 + lgst-str - 1;
   Xl = x_1;

   // 3. calculate the number of panels, to get Y2
   i = 0;
   while (str_of_menu[i][0] != '') i++; // accumulate i until no more panels
   if (i > MAX-M-PANEL) Y2 = y_1 + MAX-M-PANEL - 1;
   else Y2 = y_1 + i - 1;
   Y1 = y-1;

   // 4. check the length -- not to exceed the lower/right screen border
   if (X2 > 78 II Y2 > 23) {
     cerr << "\nCan not init. menu: border (lower/right) out of screen.\n";
     exit (-1);
   }

   // 5. get the menu-str from str_of_menu
   for (i = 0; ii < MAX-M-PANEL; i++)
     for (int j = 0; j < MAX_M_CHARS; j++)
       menu_str[i] [j] = str_of_menu[i][j]; // copy
```

```
  // 6. fill in the empty right part of each string with spaces.
  for (i = 0; i < MAX-M-PANEL; i++) {
    int j = (X2 - Xl + 1) - strlen(menu_str[i]); // panel-width = X2-X1+1
    while (j > 0) { // keep concatenates only if not yet filled
        strcat(menu_str[i], " ");
        j--;
    } // end concatenate for one panel string
  } // end for
} // end ***************************************** constructor


// show all-the panels and hilite first panel before taking any choice
void PullDown::init_show (){
  textcolor(N_TEXT_COLOR); textbackground(N_BG_COLOR);

  // draw the top border of box
  const int panel-width = X2 - X1 + 1;
  gotoxy(X1 - 1 , Y1 - 1);
  cprintf("Ú");
  for (int i = 0; i < panel-width; i++)
    cprintf("Ä");
  cprintf("¿");
  // show all the panels
  const int no_panels = Y2 - Y1 + 1;
  int y = Y1;
  for (i = 0; i < no_panels; i++) {
    gotoxy(X1 - 1, y);
    cprintf("%s", "³"); // left border
    cprintf("%s", menu_str[i]); // print using private data member
    cprintf("%s", "³"); // right border
    y++;
  } // end for
  // draw the bottom border of box
  gotoxy(X1 - 1 , Y2 + 1);
  cprintf("À");
  for (i = 0; i < panel-width; i++)
    cprintf("Ä");
  cprintf("Ù");
  // hilite the FIRST panel
  textcolor(H_TEXT_COLOR); textbackground(H_BG_COLOR); // use hilite colors
  gotoxy(X1, Y1);
  cprintf("%s", menu_str[0]); // first string
  gotoxy(X1, Y1); // put cursor back at starting of string
  // set normal colors again, for safety
  textcolor(N_TEXT_COLOR); textbackground(N_BG_COLOR);
} // end function ****************************************** init_show


// change the hilite panel from prev panel to cur panel
void PullDown::change_hlite (int prev, int cur) {
  // normal the prev.
  textcolor(N_TEXT_COLOR); textbackground(N_BG_COLOR);
  gotoxy(X1, prev + Y1);
  cprintf("%s", menu_str[prev]);
  // hilite the current
  textcolor(H_TEXT_COLOR); textbackground(H_BG_COLOR);
  gotoxy(X1, cur + Y1);
  cprintf("%s", menu-str[cur]);
  gotoxy(X1, cur + Y1); // put the cursor back at the beginning
  // set normal colors again, for safety
  textcolor(N_TEXT_COLOR); textbackground(N_BG_COLOR);
} // // end function • ⌧⌧⌧♦♦✍⌧♦⌧♦♦⌧⌧♦⌧♦▤♦⌧⌧⌧⌧⌧⌧⌧⌧⌧⌧⌧⌧⌧⌧⌧⌧✍⌧⌧ change_hlite
```

```
// get the choice from pull down menu
int PullDown::choose () {

  // allocate memory to store current screen contents
  char *scr_buf;
  scr-buf = new char[ ((X2+1) - (X1-i) + 1 ) * ( (Y2+1) -- (Y1-1) + 1 ) * 2 ];
  if (!scr_buf) {
    cerr << "Video-memory allocation error\n";
    cerr << 'in choose() in pulldown.cpp\n";
    exit (1);
  }
  // store current screen contents
  gettext(X1-1, Yl-1, X2+1, Y2+1, scr-buf);
  init_show (); // show all, hilite 1st.
  int cur_panel = 0, prev_panel;
  char ch;
  const int no-panels = Y2 - Y1 + 1;
  do {
    ch = getch();
   -switch ( ch ) {
      case DOWN_ARROW:
        prevganel  =  cur-panel;
        if (cur_panel != no-panels - 1)
          cur_panel++;
        else
          cur_panel = 0;
        change-hlite  (prevganel,  cur_panel);
        break;
      case  UP-ARROW:
        prevganel  =  cur_panel;
        if (cur_panel != 0) // panels are from 0 to (no-panels - 1)
          cur-panel--;
        else
          cur_panel = no_panels - 1;
        change-hlite (prev_panel, cur-panel);
        break;
      case  ENTER:
  //  restore  the  previous  screen  contents,  delete  the  buffer  that  holds  it
        puttext(X1-1,  Yl-1,  X2+1,  Y2+1, scr-buf);
        delete   scr-buf;
        return  (cur-panel  +  1);
      case  ESC:
  //  restore  the  previous  screen  contents,  delete  the  buffer  that  holds  it
        puttext(X1-1,  Yl-1,  X2+1,  Y2+1, scr-buf);
        delete   scr-buf;
        return  ESC;
      case   LEFT-ARROW:
  //  restore  the  previous  screen  contents,  delete the buffer that holds it
        puttext(X1-1,  Y1-1,  x2+1,  Y2+1,  scr-buf);
        delete   scr-buf;
        return   LEFT-ARROW;
      case   RIGHT-ARROW:
  //  restore  the  previous  screen  contents,  delete the buffer that holds it
        puttext(X1-1,  Y1-1, X2+1,  Y2+1,  scr-buf);
        delete   scr-buf;
        return   RIGHT-ARROW;
    ) // end switch

  } while (1);  //  end  while

} // end function ****************************************** choose
```

APPENDIX D
THE CODE OF MCQTEST.CPP

```
// ******************************************************* //
// The program-that score and analyse the multiple-choice //
// question tests items.                                  //
//                                                        //
// By: Mohamad Zamberi Saad                               //
//     For the-completion of MSc of Information Technology //
//     Completed between Jun 1995 - September 1995        //
//                                                        //
// to make executable file:                               //
// 1. make pulldown menu object file                      //
// compile : tcc -c -P pulldown.cpp                       //
//                                                        //
// 2. make executable file:                               //
// compile : tcc -P mcqtest.cpp pulldown.obj              //
//                                                        //
// ******************************************************* //

#include <iostream.h>  // needed by cout and cerr
#include <fstream.h>   // needed by ofstream
#include <conio.h>     // needed by gotoxy() -- and cprintf()
#include <stdlib.h>    // needed by exit() in main function
#include "pulldown.h"  // needed to use the pull down menu system

// *******************************************************
// MENU AND SCREEN PART

// this structure will hold information about menu titles
// and first coordinates of pull down menus
struct menu {
  char title_str[10]; // title string, ex: 'File'
  int t_leftX;        // title left X
  int pd_leftX;       // pull down left X
};

// keys on keyboard -- addition to those defined in pulldown.h
const int UP-ARROW = 72;
const int PAGE-UP = 73;
const int DOWN_ARROW = 80;
const int PAGE-DOWN = 81;
const int CTRL_HOME =  119;
const int CTRL_END =  117;
const int BACKSPACE = 8;

// menu and screen constants
const int NO_MENU = 5; // number of pull down menus
const int TITLE-Y = 1; // top co-ordinate of the menu titles
const int UPPER-Y = TITLE-Y + 2; // top co-ordinate of a pulldown menu
const int MAXCOLSCR = 80;  // maximum screen column
const int HALFCOLSCR = 38; // half of screen column
const int MAXROWSCR- = 25;  // maximum screen row
const int SCOR_CLR = CYAN;
const int ANA-CLR = BLUE;
const int TRUE = 1;
const int FALSE = 0;

// menu and screen functions
int  activ8menu (int n, menu menu-item[], PullDown cur-menu);
void draw-border(int xl, int yl, int x2, int y2, int fgcol, int bgcol);
void ready-screen ();
void tidy_up();
void show-mesg(char *err-msg, int err);
void paint-box(int xl, int yl, int x2, int y2, int col);
void use-text-color(int fgcol, int bgcol);
void use_normal_color();
void prompt-line(char *msg, char *str_toget, int len);
void statusline (int n);
void show-status (char ● linel, char ● line2);
void displayclass (char *clasname, char *testname);
int  sub_pulldown (int leftX, int topY, char streeng[][MAX_M_CHARS]);

// ******************************************************
// CORE PROGRAM THAT CALCULATE AND PRODUCE OUTPUT

int const MAXSTUDENTS = 150; // maximum students in the test
int const MAXQUESTN = 100;   // maximum questions in the test
int const MAXCNAME = 20;     // maximum characters in a student's name
int const MAXCHOICES = 5;    // maximum choices in one question

// init functions
void initanswer (char answers[][MAXQUESTN]);
void initnames (char toinit[][MAXCNAME]);
void init1Darray (char *toinit, int limit);
int  readteacansw (char *to-read, char *filename);
int readstudans (char to_read[][MAXQUESTN], char names[][MAXCNAME],
       char *filename);

// support functions
int  len_ofstr (char *tocount);
int countstudents (char answers[][MAXQUESTN]);
void copy-row (char tochange[][MAXQUESTN], char *tobelD, int row_num);
void copy_col (char tochange[][MAXQUESTN], char ● tobelD, int col_num);
int  no-correct (char *arrayA, char *arrayB);
int  no-wrong (char *arrayA, char *arrayB);
int guess-correct (int right, int wrong, int choices);
int  no-match (char to-match, char *cur-question, int start, int stop);
float round-to2 (float toround);
void get-streeng (char *to-get, int lento_get, int startX, int y);

// process functions
void scoring (char answers[][MAXQUESTN], char *teachers,
   int score[], int choices);
void do_diff_ind (char answers[][MAXQUESTN], char *teachers,
   float diff_ind[]);
int  resp_count(char to-count, char *cur-question);
void do_resp_count (char answers[][MAXQUESTN], int resp_table[] [MAXQUESTN]);
void do-disc-index (char answers[][MAXQUESTN], char *teachers,
   int score[], float disc_index[] );
void sort-answers(char answers[][MAXQUESTN], int score2[], int no-students);

int print-result (char *out-fname, char name_arr[] [MAXCNAME], int score[],
   int no-students, int no-questions, float disc-index[], float diff_index[],
   int resp[][MAXQUESTN], char *cl-name, char *subj_name, int res2print);
// displaying
void doshowscore (char names[][MAXCNAME], int score[],
   int no-students, int no-qtns);
void showscore (char names[][MAXCNAME], int score[], int top,
   int bottom, int no-qtns);
void doshowanalysis (float disc_index[], float diff_index[],
     int resp[][MAXQUESTN], int no-questions, char *teachersanswers);
void showanalysis (float disc_index[], float diff_index[],
     int resp[][MAXQUESTN], int top, int bottom, char *teachersanswers);

void load-data(char answers[][MAXQUESTN], char names[][MAXCNAME],
  char    *teachers-answers);
void show-settings (int choices, char *clasname, char *testname,
  int no-q, int  no-students);

// flags
int fl_dataloaded = FALSE;
int fl_usegc = FALSE;
int fl_count_perc = FALSE;
int fl_saved = FALSE;

// ************************************************************

main () {

  char answers[MAXSTUDENTS][MAXQUESTN]; // hold the students' answers
  char teachanswer[MAXQUESTN];          // hold the teacher's answers
  char names[MAXSTUDENTS][MAXCNAME];    // students' names
  int  no-questions;           // # of questions
  int qchoices = 5;            // # of choices in a question
  int  score[MAXSTUDENTS];     // hold the scores

  float disc_index[MAXQUESTN];
  float diff_index[MAXQUESTN];
  int resp_table[MAXCHOICES+1][MAXQUESTN];
  // + 1 to accommodate invalid answers or omittance.
  char *clasname = "                    "; // temporarily
  char *testname = "                    "; // temporarily

  // ************************************************************
  // MENU PART

  // declare an array of menu titles
  menu menu_arr[] = {
    ("File',       5,  5,},
    {"Result',    12, 12,},
    {"Scoring", 22,  22,},
    ('Analysis", 33, 33,},
    {'Other",     45, 45,},
  };

  ready_screen();

  for (int i = 0; i < NO-MENU; i++) {      // print pd menu titles
    gotoxy(menu_arr[i].t_leftX, TITLE-Y);
    cprintf("%s",menu_arr[i].title_str);
  }

  char filetext[MAX_M_PANEL][MAX_M_CHARS] = {
    'Load, score and analyse a test file", "Quit this program'
  };

  char resulttext[MAX_M_PANEL][MAX_M_CHARS] = {
    *Browse scores (on the screen)', "Browse analysis result (on screen)',
    "Send results to output file'
  };

  char scoretext[MAX_M_PANEL][MAX_M_CHARS] = {
    "Rescore, NOT using guessing correction',
    "Rescore, USE guessing correction',
    "Change number of choices in a question.,
  };

  char anlystext[MAX_M_PANEL][MAX_M_CHARS] = {
    *Report response count in %",
  'Report  resp. count in raw numbers".,
};

char othertext[MAX_M_PANEL][MAX_M_CHARS] = {
  'Display  all  current  settings',
  "Type in or change CLASS name',
  "Type in or change TEST name',
};

// declare the pull down menus, one by one
PullDown filemenu   (menu_arr[0].pd_leftX, UPPER-Y, filetext);
PullDown resultmenu (menu_arr[1].pd_leftX, UPPER-Y, resulttext);
PullDown Scoremenu  (menu_arr[2].pd_leftX, UPPER-Y, scoretext);
PullDown anlysmenu  (menu_arr[3] .pd_leftX, UPPER-Y, anlystext);
PullDown othermenu  (menu_arr[4].pd_leftX, UPPER-Y, othertext);

// declare an array that store all of the pulldown menus
PullDown pdown[NO_MENU] = {
  filemenu,  resultmenu,  scoremenu,  anlysmenu,  othermenu,
};
PullDown *pd_pointer;

// navigate through all menus here
int choice = 0, cur-menu = 0;
do {
  pdgointer = &pdown[cur_menu]; // obtain pointer to 'cur_menu'th element
  statusline(1);
  choice = activ8menu (cur-menu, menu-arr, *pd_pointer); // activate

  switch ( choice ) {

    case  RIGHT-ARROW:
      if (cur-menu != NO-MENU - 1) cur-menu++; else cur-menu = 0;
      break;

    case  LEFT-ARROW:
      if  (cur-menu != 0) cur-menu--; else cur-menu = NO-MENU - 1;
      break;

  } // end switch

  // TAKE ACTIONS HERE.
  // check first, whether data has been loaded
  if (!fl_dataloaded && choice != RIGHT-ARROW && choice != LEFT-ARROW
    && cur-menu > 0 )
    show_mesg("You must open a test data file first!', 1);
  else { // *** 1 ***
  switch (cur-menu) {

    case 0: // 'File menu'
      switch (choice) {
        case 1:  // load input file
          // make sure: no other file opened but not been saved
          int oktoload = (!fl_dataloaded) I I (fl_dataloaded && fl_saved);
          if (!oktoload) {
            show-mesg ('Results of THIS test has not been saved yet.', 1);
            char menutext[MAX_M_PANEL] [MAX-M-CHARS] = {
              'Cancel, do not load a new test file',
              'Yes, load and do NOT save this one',
            };
            int Xleft = 28, Ytop = 12;
            int decide = subgulldown (Xleft, Ytop, menutext);
            if (decide == 2) oktoload = TRUE;
          }
```

```
            if (oktoload) {
            statusline(4);
            load-data(answers,   names,   teachanswer);
3           init1Darray(clasname, 20);
            init1Darray(testname, 20);
            displayclass(clasname, testname);
            // re-init areas of results display
            paint-box (1, TITLE-Y + 3, HALFCOLSCR, MAXROWSCR - 2, SCOR_CLR);
            paint-box  (HALFCOLSCR  +-1,  TITLE-Y  +  3,
              MAXCOLSCR, MAXROWSCR - 2, ANA_CLR);
            // score and analysis, automatically
            scoring  (answers,  teachanswer,  score,  qchoices);
            do_diff_ind (answers, teachanswer, diff_index);
            do_resp_count (answers, resp_table);
            do-disc-index  (answers,  teachanswer,  score,  disc-index);
            fl-saved  =  FALSE;
            } // end if oktoload
            break;

          case 2:  // quit program
            int oktoexit = (!fl_dataloaded) II (fl_dataloaded && fl-saved);
            if (!oktoexit) {
              show-mesg  ("Results  has  not  been  saved  yet.',  1);
              char menutext[MAX_M_PANEL][MAX_M_CHARS] = {
                "Cancel,  do  not  quit',
                'Yes,  quit  and  do  not  save*,
              };
              int Xleft = 30, Ytop  = 12;
              int decide = sub_pulldown (Xleft, Ytop, menutext);
              if (decide == 2) oktoexit = TRUE;
            } // end if
            if (oktoexit) {
              // the only NORMAL PROGRAM EXITING is through here
              tidy_up();
              cout << 'Program  ends  OK\n";
              exit  (1);
            }

        } // end switch (choice) in cur-menu = 0;
        break; // case cur-menu == 0

      case 2: // 'Scoring menu"
        // NO need of using switch (choice) here.
        if (choice == 1 I I choice == 2) { // prevent activate with LEFT/RIGHT
            if (choice == 1) fl_usegc = FALSE;
            else fl_usegc = TRUE;
            scoring  (answers,  teachanswer,  score,  qchoices);
            statusline(2);
            doshowscore  (names,  score,
              countstudents(answers),    len_ofstr(teachanswer));
        }
        else  if  (choice  ==  3 )  (  // change # of alternatives
          // give pull down menu of 2,3,4,5 number of choices.
          statusline(8);
          show-mesg  ('How  many  choices  in  a  question?',  0);
          char  menutext[MAX_M_PANEL] [MAX-M-CHARS]  =  {
            "2  choices  --  ex:  True/False',
            "3 choices -- ex: A,B,C",
            "4 choices -- ex: A,B,C,D",
            "5 choices -- ex: A,B,C,D,E",
          };
          int  Xleft  =  30,  Ytop  =  12;
          const int oldc = qchoices; // caution : if press ESC.
          qchoices =  (1  +  sub_pulldown  (Xleft,  Ytop,  menutext));
          if (qchoices == ESC + 1 || qchoices == LEFT-ARROW + 1
             II qchoices == RIGHT-ARROW + 1) qchoices = oldc;
        } // end if
        break; // case cur-menu == 1

      case 3: // "Analysis menu'
        switch (choice) {
          case 1: // analysis, showing resp. count in %
          fl_count_perc  =  TRUE;
            doshowanalysis   (disc-index,   diff-index,   resp-table,
              len_ofstr(teachanswer), teachanswer);
            break;
          case 2: // analysis, NOT showing resp. count in %
            fl_count_perc = FALSE;
            doshowanalysis   (disc-index,   diff-index,   resp-table,
              len_ofstr(teachanswer), teachanswer);
            break;
        } // end switch (choice) in cur_menu = 2;
        break; // case cur-menu == 2

    case  1:  //  "Result  menu"
        switch (choice) {
          case  1:
            statusline(2);
            doshowscore   (names,  score,
              countstudents(answers),    len_ofstr(teachanswer));
            break;

          case  2:
            statusline(3);
            doshowanalysis   (disc-index,   diff-index,   resp-table,
              len_ofstr(teachanswer), teachanswer);
            break;

          case 3: // write to output file
            char menutext[MAX_M_PANEL] [MAX_M_CHARS] = {
              'Write  SCORES  only",
              "Write  ITEM  ANALYSIS  RESULTS  only',
              "Write  BOTH  of  the  above",
            };
            int Xleft  = 28,  Ytop  =  12;
            int res2print = sub_pulldown (Xleft, Ytop, menutext);
            if (res2print >= 1 && res2print <= 3) {

              char  *outfile_name = "             ";
              prompt-line  ('Please  type  in  the  OUTPUT  filename:',
                outfile_name, 12);
              if  (print-result  (outfile-name,  names,  score,
                countstudents(answers),    len_ofstr(teachanswer), disc-index,
                diff-index, resp-table, clasname, testname, res2print)
              ) { // succesfully written.
                show-mesg  ("OK.  Report  written  in  that  file',  0);
                fl_saved = TRUE;
              }
              else-  show-mesg  ('Sorry.  Can  not  open  that  OUTPUT  file',  1);

            } // end if
            break;

        } // end switch (choice) in cur-menu = 3;
        break; // case cur-menu == 3

      case 4: // 'Options menu"
        switch (choice) {
            case 1: // show current settings
              statusline(7);
              show-settings (qchoices, clasname, testname,
                len_ofstr(teachanswer), countstudents(answers));
              break;
            case 2: // input class name
              statusline(5);
              prompt-line('Please type in the CLASS name:., clasname, 20);
              displayclass(clasname, testname);
              break;
            case 3: // input test name
             -statusline(6);
              prompt_line("Please type in the TEST name:", testname, 20);
              displayclass(clasname, testname);
              break;

          } // end switch (choice) in cur-menu = 4;
          break; // case cur-menu == 4

      } // end switch (cur-menu)
      } // end *** 1 ***

    } while (1); // end while

  } // end main

  // ***************************************************************
  // second level functions -- called from the main() only -- by menu

  // load data from ASCII text file into answers, names & teachers-answers
  void load_data(char answers[] [MAXQUESTN],char names[] [MAXCNAME],
    char *teachers_answers) {

    // read in the INPUT file name
    char *infile_name = "             ● ; // temporarily
    // to avoid 'Null pointer assignment'
    prompt_line("Please type in the test data filename:., infile_name, 12);

    init1Darray(teachers_answers, MAXQUESTN);
    if (readteacansw(teachers_answers, infile_name)) {
    // if succesfully read teachers answers
      initnames(names); // proceed, reading students answer
      initanswer(answers);
      if (readstudans(answers, names, infile_name)) {
      // if successfully read students answer
        fl-dataloaded = TRUE;
        show_mesg("OK. Choose Browse from Result menu to see the result:, 0);
      } // end if readstudans
      else { // if NOT succeed in readstudans
        show-mesg ('Error in reading student answers", 1);
        initnames(names);
        initanswer(answers);
        fl-dataloaded = FALSE; // to prevent further options
      } // end else.-- end of reading students answer
    } // end if readteacansw
    else ( // failed reading teachers answers
      fl-dataloaded = FALSE;
      show-mesg ('Error: Can not open that INPUT file", 1);
    } // end else

  } // end function ************************* load_data
```

```
// ***********************************************************
// CORE PROGRAM THAT CALCULATE AND PRODUCE OUTPUT

// this function will open the output file, then write the score
// and the results of item analysis
int print_result (char *out_fname, char name_arr [ I [MAXCNAME], int score [],
  int no_students, int no_questions, float disc_index[], float diff_index[],
  int resp[][MAXQUESTN], char *cl_name, char ● ♋♌♏♐⌧■♋○♏⌧ int res2print){

  ofstream  file-out  (out-fname);
  if (!file_out) // can't open file
    return 0;

  // print test information
  file-out << "                Class:  . << cl-name << "\n";
  file-out << "                 Test:  " << subj_name << "\n";
  file-out << .  Number of students: " << no-students << "\n";
  file-out << " Number of questions: .<< no-questions << "\n\n";

  if (res2print != 2) { // if not only printing item analysis results
    // show names and scores
  file-out  <<  "\t\t\tTEST RESULT\n";
    file-out << " Names                    .   *Scores* << .  Percentage\n";
    file-out << "-----------------------------------------------------";
    for (int stud = 0; stud < no-students; stud++) {
      // file-out << stud + 1;
      for (int j = 0; j < MAXCNAME; j++)
        file-out << name_arr[stud][j];
      file-out << "\t";

      file-out << score[stud] << "\t"
        << round_to2((score[stud]/(no_questions+0.00))*100); // percentage
    ) // end printing all students
    file-out << "\n\n";
  } // end if

  if (res2print != 1) { // if not only printing scores
    // show discrimination index and difficulty index
    file-out << "\t\t\tITEM ANALYSIS RESULTS\n";
    file-out << "\tDiscr.\tDiff.\t\tResponse count\n";
    file-out << "Q#\tindex\tindex\tA\tB\tC\tD\tE\t\-\n";
    file-out
    << "------------------------------------------------------------\n";
    for (int ques = 1; ques <= no_questions; ques++) {

      file-out << (ques) << "\t"
        << round_to2(disc_index[ques - 11) << "\t";        // disc ind
      file-out << round_to2(diff_index[ques - 1]) << "\t";  // diff ind

      for (int row = 0; row < (MAXCHOICES+1); row++) {      // resp count
        file-out << resp[row][ques - 11;
        if (fl_count_perc) {
          const int perc = resp[row][ques - 11 / (0.00 + no-students) * 100;
          file-out << "(" << perc << ")";
        }
        file-out << "\t";
      } // end printing resp. count for each question

      file-out << "\n";

    } // end for -- end printing all ana. results for each question
    file-out << "\n";

  } // end if
  file_out.close();
  return 1;

} // end function ************************* print_result


// initialize the answer 'array, filling all with '' char.
void initanswer(char answers[] [MAXQUESTN]) {
  for (int i = 0; i < MAXSTUDENTS; i++)
    for (int j = 0; j < MAXQUESTN; j++)
      answers[i][j]= '';
} // end function ************************* initanswer


// initialize the names array, filling all with '' char.
void initnames(char toinit[] [MAXCNAME]) {
  for (int i = 0; i < MAXSTUDENTS; i++)
    for (int j = 0; j < MAXCNAME; j++)
   toinit[i][j]=  '  ';
} // end function • ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠ initnames


// read in students answer, line by line.
int readstudans(char to_read[] [MAXQUESTN], char names [][MAXCNAME],
      char *filename) {

  ifstream file-in (filename);
  if (!file_in) {
    return 0;
  I

  // 1. to ignore the 1st data, i.e. teachers answer.
  // 2. and to count the number of questions
  char  *dummy;
  file-in >> dununy;
  const int no-quest = len_ofstr(dummy);

  int i;
  for (int t = 0; t < MAXSTUDENTS; t++){

    ı = 0;                          // read names -- char by char
    char c;
    while (file-in.get(c), c != '*' && !file_in.eof()) {
      names[t][i] = c;
      i++;
    } // end of reading the name, indicated by an '*'.
    if (!c) { // if c is eof() -- Ctrl-Z
        file-in.close();
        return 0;
    }

    file-in >> to_read[t]; // read answers for one student -- one string
    if (len_ofstr(to_read[t]) != no-quest && to_read[t] [0] != '') {
      show_mesg("Number of student's answer is not same as teacher's', 1);
      return 0;
    }
    if (!to_read[t][0]) // Ctrl-Z in answers.
      break;

  } // end for

  // close input file
```

```
    file_in.close();
    return 1; // succeed in reading

} // end function ************************* readstudans


  // initialise 1D array
  void init1Darray(char ● toinit, int limit) {
    for (int j = 0; j < limit; j++)
      toinit[j]= '';
  } // end function ************************* init1Darray


  // read the teachers answer, return succeed or not
  int readteacansw(char *to-read, char *filename) {

    ifstream file-in (filename);
    if (!file_in) // can not open INPUT file
      return 0;

    file-in >> to-read;     // READ FIRST LINE ONLY

    file-in.close();
    return 1;

  } // end function • ************************ readteacansw


  // count how many questions (or the length of the string)
  int len_ofstr(char *tocount) {
    int res = 0;
    char *p = tocount;
    while (*p++ != '')
      res++;
    return res;
  ) // end function ************************* len_ofstr


  // given the student answers array, return the number of students
  int countstudents(char answers[] [MAXQUESTN]) {
    int res = 0;
    while (answers[res][0] != '' && res < MAXSTUDENTS)
      res++;
    return res;
  } // end function ************************* countotudento


  // scores the students answer, by comparing one student each time.
  void scoring (char answers[] [MAXQUESTN], char *teachers,
    int score[], int choices) {

    // for using guessing correction
    int wrong' = 0;

    int row = 0;
    char cur_student[MAXQUESTN];

    while (answers[row][0] != '') {
      init1Darray(cur_student, MAXQUESTN);
      copy-row(answers, cur-student, row);
      score[row] = no_correct(teachers, cur-student);
      if (fl_usegc) ( // use guessing correction or not
        wrong = no_wrong(teachers, cur-student);
```

```
            score[row] = guess-correct(score[row], wrong, choices);
            if (score[row] < 0) score[row] = 0;
          } // end if
          row++;
        ) // end while

} // end function • ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠ scoring


// calculate the actual score from right and wrong answers.
int guessjcorrect (int right, int wrong, int choices) {
  int  result;
  result = right - (wrong/(choices - 1)); // make sure choices not < 2
  return  result;
} // end function ************************* guess_correct


// extract, one row from 2D array to a 1D array.
void copy-row (char tochange[] [MAXQUESTN], char *tobe1D, int row_num) {
  for (int i = 0; i < MAXQUESTN; i++)
    tobe1D[i] = tochange[row_num] [i];
} // end function ************************* copy-row


// extract one row from 2D array to a 1D array.
void copy_col (char tochange[] [MAXQUESTN], char *tobe1D, int col_num) (
  for (int i = 0;  i  < MAXSTUDENTS;  i++)
    tobe1D[i] = tochange[i][col_num];
} // end function ************************* copy_col


// return how many matching between 2 1D arrays, excluding ''.
int no-correct (char *arrayA, char ●  arrayB) {
  int i = 0, result  = 0;
  while (arrayA[i] != '') {
    if (arrayA[i]  == arrayB[i])
      result++;
    i++;
  }
  return  result;
} // end function ************************* no_correct


// return how many wrong answers between 2 1D arrays, excluding ''.
int no-wrong (char *arrayA, char ●  arrayB) {
  int i = 0,  result = 0;
  while  (arrayA[i] != '') {
    if (arrayA[i] != arrayB[i]  && arrayB[i] != '-')
      result++;
    i++;
  }
  return  result;
) // end function ************************* no_wrong


// do calculate diff. index
void do_diff_ind (char answers[][MAXQUESTN], char *teacher@,
  float diff_ind[]) {

  int col = 0, right = 0,  no-students = 0;
  no-students = countstudents(answers);
  char cur_question[MAXSTUDENTS];
  while (answers[0][col] != '') {
      init1Darray(cur_question, MAXSTUDENTS); // for safety
      copy-col(answers, cur-question, col);
      right = resp_count(teachers[col], cur-question);
      diff_ind[col] = (right + 0.00) /-no-students;
    col++;

   } // end while
} // end function ************************ do diff ind


// return how many occurance of to-count in cur-question array, excluding ''.
int resp_count(char to-count, char *cur_question) {
  int i = 0, result = 0;
  while (cur_question[i] != '') {
    if (cur-question[il == to-count)
      result++;
    i++;
  }
  return result;
} // end function * ************************ resp count

// do the response counting
void do-reap-count (char answers[][MAXQUESTN], int resp_table[][MAXQUESTN]) {
  int col = 0;
  char cur_question[MAXSTUDENTS];
  while (answers[0][col] != '') {
    init1Darray(cur_question, MAXSTUDENTS); // for safety
    copy-col(answers, cur-question, col);   // for simplicity
    resp_table[0][col] = resp_count('A', cur-question);
    resp_table[1] [col] = resp_count('B', cur-question);
    resp_table[2][col] = resp_count('C', cur-question);
    resp_table[3] [col] = resp_count('D', cur-question);
    resp_table[4][col] = resp_count('E', cur-question);
    resp_table[5] [col] = resp_count('-', cur-question);
    col++;
  } // end while
} // end function • ************************ do-reap-count


// do the disc. Index calculation
void do-disc-index (char answers[][MAXQUESTN], char *teachers,
  int score[], float disc_index[ ]) {

  // make a new copy of answers
  char new_answers[MAXSTUDENTS][MAXQUESTN];
  for (int i = 0; i < MAXSTUDENTS; i++)
    for (int j = 0; j < MAXQUESTN; j++)
      new_answers[i] [j] = answers[i][j];  // copy

  // make a new copy of scores
  int new_score[MAXSTUDENTS];
  for (i = 0; i < MAXSTUDENTS; i++)
    new_score[i] = score[i];     // copy

  const int no-studs = countstudents(answers);
  sort-answers (new-answers, new-score, no-studs);

  // calculate the disc. index on this new sorted answers

  const float no-students = no-studs + 0.00;
  // make it float for further calculation.
```

```
    const float margin = 0.30;
    const int gap = (no_students * margin);
    const int top-start = 0;
    const int top-stop = gap - 1; // 30% highest
    const int bot_start = no-students - gap; // 30% lowest
    const int bot_stop = no-students - 1;

    float pH = 0.00, pL = 0.00;
    int col = 0;
    const float gap-float = gap + 0.00;
    char    cur-question[MAXSTUDENTS];
    while (new-answers[O][col] != '') {
      init1Darray(cur_question, MAXSTUDENTS); // for safety
      copy_col(new_answers, cur-question, col);
      pH = no-match (teachers[col], cur-question,
        top-start, top-stop) / ( gap-float );
      pL = no-match (teachers[col], cur-question,
        bot_start, bot_stop) / ( gap-float );
      disc_index[col] = pH - pL;
      col++;
    } // end while

  } // end function ************************* do_disc_index


  // sort answers descendingly, before do disc. index
  void sort_answers(char answers[] [MAXQUESTN], int score2[], int no_students) {

    char swap[MAXQUESTN];
    for ( int i = 0; i < no-students; i++) (      //  sorting.
      for ( int j = 0; j < i; j++) {
        if (score2[j] < score2[i]) (  // descending
          // swap the answers of the students,

          //1. copy-row i into a swap
          copy-row (answers, swap, i);

          //2. copy row j into row i
          for ( int k = 0; k < MAXQUESTN; k++ )
            answers[i][k] = answers[j][k];

          //3. copy swap into row j
          for ( k = 0; k < MAXQUESTN; k++ )
            answers[j][k] = swap[k];

          // AND sort the score, also.
          const int swap = score2[i];
          score2[i] = score2[j];
          score2[j] = swap;
        } // end if
      } // end for j
    } // end for i -- end of sorting

  } // end function ************************* sort_answers


  // return how many occurrence of to-match in cur-question array.
  int no_match (char to-match, char *cur_question, int start, int stop) {

    int result = 0;
    // note: use <= instead of < , because want to include stop margin
    for (int i = start; i <= stop; i++)
      if ( cur-question[i] == to-match )
        result++;
    return   result;

  } // end function ************************* no_match


  // call the function that will show the score
  void doshowscore (char names[][MAXCNAME], int score[],
    int no-students, int no_qtns) {

    const int upper-screen = TITLE-Y + 3;
    const int lower-screen = 23; // add this to add screen length
    const int max_element = no-students - 1; // because start from element 0
    int tall = lower-screen  -  (upper-screen);
    if (tall > no-students) tall = no-students;

    // top & bottom store the element number to display,
    // from top to bottom. bottom is bigger than top.
    int top = 0;  // start with first element in the array
    int bottom = tall - 1; // if tall = 5, this is 4, so do from 0 to 4 .

    use_text_color(YELLOW, SCOR-CLR);
    gotoxy(1, upper-screen);
    cprintf("%s", " #      Name          Score   (%)");

    use-text-color(WHITE,    SCOR-CLR);
    window (1, upper-screen t 1, HALFCOLSCR, upper-screen t tall t 1);
    // need to use the scoring result area as the current window,
    // because the showscore() will print relative to that area.

    char ch;
    do {

      // extreme case on the low end of array -- i.e. top of display
      if (top < 0) {
        top = 0;
        bottom = tall - 1; // like early state
      }

      // extreme case on the high end of array -- i.e. bottom of display
      if (bottom > max-element) {
        bottom = max-element;
        top = max-element - tall t 1;
      }

      // display from 'top'th el. to 'bottom'th el.
      showscore (names, score, top, bottom, no-qtns);

      ch = getch();

      switch ( ch ) {

        case PAGE-DOWN: // increase the margins by the length of one screen
          top += tall; bottom += tall; break;
        case PAGE_UP: // decrease the margins by the length of one screen
          top -= tall; bottom -= tall; break;
        case DOWN_ARROW: // increment the margins
          top++; bottom++; break;
        case UP-ARROW: // decrement the margins
          top--; bottom--;  break;
        case CTRL_HOME: // like early state
          top = 0; bottom = tall - 1; break;
        case CTRL_END: // go to far end
          bottom = max-element; top = max-element - tall t 1; break:
        case ESC:
          use_normal_color();
          window (1, 1, MAXCOLSCR, MAXROWSCR); // set back to normal
          return;

      } // end switch

    } while (1); // end while

} // end function * ************************ doshowscore

// display scores from top to bottom
void showscore (char names[][MAXCNAME], int score[], int top,
  int bottom, int no_qtns) {

  const int left = 4; // first char of a name will appear here
  int y = 1; // start from top co-ord of CURRENT window
  for (int i = top; i <= bottom ; i++) { // print for each student

    use-text-color(YELLOW, SCOR-CLR);
    gotoxy(left - 3, y); // print students number
    cprintf("%3d", i + 1);
    use-text-color(BLACK, SCOR-CLR);

    for (int j = 0; j <= MAXCNAME; j++) { // print name
      gotoxy(left + j, y);
      cprintf("%c", names[i][j]);
    }
    gotoxy(left + MAXCNAME + 2, y); // print score
    cprintf("%3d", score[i]);

    gotoxy(left + MAXCNAME + 7, y); // print score in %
    cprintf("%3.0f", (score[i] / (no-qtns + 0.00)) * 100);

    y++;
  } // end print for each student

} // end function * ************************ showscore


// call the function that will show the analysis results
void doshowanalyoia (float disc_index[], float diff_index[],
    int resp[][MAXQUESTN], int no-questions, char * teacheroanowers) {

  const int upper-screen = TITLE-Y + 3;
  const int lower-screen = 23; // add this to add screen length
  const int max_element = no-questions - 1; // because start from element 0
  int tall = lower-screen - (upper-screen);
  if (tall > no-questions) tall = no-questions;

  int top = 0;  // start with first element in the array
  int bottom = tall - 1; // if tall = 5, this is 4, so do from 0 to 4 .

  use_text_color(YELLOW, ANA-CLR);
  gotoxy(HALFCOLSCR + 1, upper-screen);
  cprintf("%s", " Q# Disc Diff  A  B  C  D  E  -");

  use-text-color(WHITE, ANA-CLR);
  window (HALFCOLSCR + 1, upper-screen + 1, MAXCOLSCR,
      upper-screen + tall + 1);

  char ch;
  do {

    if (top < 0) {
        top = 0;
        bottom = tall - 1; // like early state
      }

      if (bottom > max-element). {
        bottom = mm-element;
        top = max-element - tall + 1;
      }

      showanalysis (disc-index, diff_index, resp,
        top,  bottom,  teachersanswers);

      ch = getch();

      switch ( ch ) {

        case PAGE-DOWN: // increase the margins by the length of one screen
          top += tall; bottom += tall; break;

        case  PAGE-UP:  // decrease the margins by the length of one screen
          top -= tall; bottom -= tall; break;
        case DOWN-ARROW: // increment the margins
          top++;  bottom++;  break;
        case UP-ARROW: // decrement the margins
          top--;  bottom--;  break;
        case CTRL_HOME: // like early state
          top = 0; bottom = tall - 1; break;
        case CTRL_END: // go to far end
          bottom = max-element; top = max_element - tall + 1; break;
        case  ESC:
          use_normal_color();
          window  (1,  1,  MAXCOLSCR,  MAXROWSCR);  // set back to normal
          return;

      } // end switch

    } while (1);  // end while

} // end function • ************************ doshowanalysis

// display analysis results
void showanalysis (float disc_index[], float diff_index[],
      int resp [] [MAXQUESTN] , int top, int bottom, char • teacheroanawera) {

    const int left = 1;
    // the right most digit of q. number will appear here, (relative)
    int y = 1; // start from top co-ord of CURRENT window

    // count the total number of students, from the number
    // of students that answer each choices, (totalling) because does not
    // want to count from their 2D answer array (not available here)
    int no-studs = 0;
    for (int choice = 0; choice < (MAXCHOICES + 1); choice++)
      no-studs += resp[choice] [top];

    for (int i = top; i <= bottom ;  i++) { // for each question

      use-text-color(YELLOW,     ANA-CLR);
      gotoxy(left, y);       // print question number
      cprintf("%3d", i + 1);
      use-text-color(WHITE,     ANA-CLR);

      gotoxy(left + 4, y);  // print disc. index
      cprintf("%5.2f", disc_index[i]);
      gotoxy(left + 11, yl;   // print diff. index
      cprintf("%4.2f", diff_index[i]); // no -ve value.

      // display resp. count
      for (int choice.= 0; choice < (MAXCHOICES + 1); choice++) (
        if (teachersanswers[i] == choice + 65)
          use_text_color(LIGHTCYAN, ANA_CLR);
        gotoxy(left + 17 + (4. * choice), y);
        if (fl_count_perc) // display in %
          cprintf("%3.0f", (resp[choice] [i]/ (no-studs + 0.00)) * 100);
        else
          cprintf("%3d", resp[choice][i]); // no -ve value.
        if (teachersanswers[i] == choice + 65) use_text_color(WHITE, ANA_CLR);
      }
      y++;

    } // end printing out for one question

} // end function ************************* showanalysis



// ****************************************************************
// MENU AND SCREEN PART

// draw border from x1,y1 to x2,y2 in fgcol
void draw_border(int xl, int yl, int x2, int y2, int fgcol, int bgcol) {

  use-text-color  (fgcol,  bgcol);

  // upper  border
  gotoxy(x1 , y1);
  cprintf("Õ");    // upper left
  for (int x = xl + 1; x <= x2 - 1 ; xtt) (
    gotoxy (x,y1);
    cprintf("Í");
  }
  cprintf("¸"); // upper right

  // left and right border
  for (int y = yl t 1; y <= y2 - 1; ytt) {
    gotoxy(x1, Y);
    cprintf("³ "); // left border

    gotoxy(x2 - 1, y);
    cprintf(" ³"); // right border
  }

  // draw the bottom border of box
  gotoxy(x1 , y2);
  cprintf("Ô");
  for (x = xl t 1; x <= x2 - 1 ; x++) {
    gotoxy(x, y2);
    cprintf ("Í");
  }
  cprintf("¾");

  use_normal_color();

} // end function ************************* draw_border


// activate the 'n'th menu in menu_item[] array,
// and taking choice from cur-menu pull down menu.
int activ8menu (int n, menu menu_item[], PullDown cur-menu) {

 gotoxy (menu_item[n].t_leftX,  TITLE-Y); // to hilite menu title
   use-text-color (H-TEXT-COLOR, H_BG_COLOR);
   cprintf("%s",menu-item[n].title-str);      // print title-in hlite

   int res = cur_menu.choose();             // take choice, from choose()

   gotoxy (menu_item[n].t_leftX, TITLE-Y); // to normal again the-title

   use_normal_color();
   cprintf("%s",menu_item[n].title_str);    // print title in normal

   return res;

} // end function ************************* activ8menu


// last thing to be called when quitting
void tidy-up () {
   show_mesg("Thanks for using this program.", 0);
   textmode(LASTMODE);
   clrscr();
} // end function ************************* tidy_up


// make the screen ready, so that ready to use by any other parts
void ready-screen 0 {

   textmode(3);  // set screen mode to 80 columns by 25 rows, color
   // paint the menu title line
   paint-box (1, TITLE-Y, MAXCOLSCR, TITLE-Y, N_BG_COLOR);

   // paint the class and test title line
   displayclass ("NO CLASS NAME","NO TEST NAME');

   // title line of score window and item analysis results window
   int y = TITLE-Y + 2;
   int bgcol = RED; // have to use to make it same as 'paint func.' color
   paint-box (1, y, MAXCOLSCR, y, bgcol);
   use-text-color(LIGHTGREEN,    bgcol);
   gotoxy(lO, y); cprintf("%s", "SCORES');
   gotoxy(HALFCOLSCR + 14, y); cprintf("%s", 'ITEM ANALYSIS RESULTS');

   // areas of results display
   // scores area
   paint-box (1, TITLE-Y + 3, HALFCOLSCR, MAXROWSCR - 2, SCOR_CLR);
   // analysis results area
   paint-box (HALFCOLSCR + 1, TITLE-Y + 3, MAXCOLSCR, MAXROWSCR - 2, ANA-CLR);

   use_normal_color();

} // end function • ************************ ready-screen


// err is used to determine the appearance of error box
void show_mesg(char *err_msg, int err) {

   int winwidth = len_ofstr(err_msg) + 2; // extra 2 on right, 2 on left
   if (winwidth > MAXCOLSCR ) { // for programmer's use only
     cerr << 'Error message exceeded screen width.';
     exit (1);
   }
```

```
  int to_pad = FALSE;
  if (winwidth % 2 != 0) {  // if odd
    winwidth++; // make it even
    to_pad = TRUE;
  }

  const int leftX = (MAXCOLSCR / 2) - (winwidth / 2);
  int rightX = (MAXCOLSCR / 2) + (winwidth / 2);

  if (!to_pad) rightX++;

  const topY = 13;
  const lowY = 15;

  char *scrn_buf;
  scrn-buf = new char[ (rightX - leftX + 1 ) * ( lowY - topY + 1 ) * 2 ];
  gettext(leftX, topY, rightX, lowY, scrn-buf);
  // for shadow
  char *scrn_buf2;  // right shadow
  scrn_buf2 = new char[ (1) *  ( lowY - topY + 1 ) * 2 ];
  char *scrn_buf3;  // bottom shadow
  scrn_buf3 = new char[ (rightX - leftX + 1 ) * (1) * 2 ];

  // for shadow
  gettext(rightX + 1, topY + 1, rightX + 1, lowY + 1, scrn_buf2);
  gettext(leftX + 1, lowY + 1, rightX + 1, lowY + 1, scrn_buf3);
  paint-box(rightX + 1, topY + 1, rightX + 1, lowY + 1, BLACK);
  paint-box(leftX + 1, lowY + 1, rightX + 1, lowY + 1, BLACK);

  if (err) draw-border (leftX, topY, rightX, lowY, LIGHTCYAN, RED);
  else draw-border (leftX, topY, rightX, lowY, BLACK,  WHITE);
  gotoxy(leftX + 2, topY + 1); // print the message
  cprintf("%s", err-msg);
  gotoxy(leftX + 2, topY + 1);

  getch();

  // restore the previous screen contents, delete the buffer that holds it
  puttext(leftX, topY, rightX, lowY, scrn-buf);
  delete  scrn-buf;
  puttext(rightX + 1, topY + 1, rightX + 1, lowY + 1, scrn_buf2);
  delete scrn_buf2;
  puttext(leftX + 1, lowY + 1, rightX + 1, lowY + 1, scrn_buf3);
  delete scrn_buf3;

  use-normal-color{};

) // and function ************************* show_mesg


// paint the area from x1,y1 to x2, y2 in 'col' color.
void paint_box(int xl, int yl, int x2, int y2, int col) {

  use_text_color(col, col); // set both text and bg to have same color

  window(xl, yl, x2, y2);
  clrscr();

  // set back to normal
  window(1, 1, MAXCOLSCR, 25);
  use_normal_color();

} // end function * ************************* □♋♓■♦♏♌□⊠
```

```
// change default text color and text back ground color
void use-text-colqr(int fgcol, int bgcol) {

  textcolor(fgcol);      // set text to fgcol
  textbackground(bgcol); // set background to bgcol

} // end function ************************* use_text_color


// set the colors back to normal
void use_normal_color() {

  textcolor(N_TEXT_COLOR);     // set text back to normal color
  textbackground(N_BG_COLOR); // set bg back to normal color

} // end function • *********************** use-normal-color


// take a string from keyboard, giving prompt in a pop up window
// restriction: str_toget must have been declared using specific style,
// ex: char *filename = "THISDATA.TST";// to avoid 'Null pointer assignment'
// then call using: prompt_line("Please type in the filename', filename, 12);
// this is because we are using user-defined function len_ofstr()
// to count the 'msg' string length
void prompt_line(char ● rasg, char ● str-toget, int len) {

   int winwidth = len_ofstr(msg) + len + 2; // extra 2 on right, 2 on left
   if (winwidth > MAXCOLSCR ) { // for programmer's use only
     cerr << "Error message exceeded screen width.";
     exit (1);
   }

   int to_pad = FALSE;
   if (winwidth % 2 != 0) { // if odd
     winwidth++; // make it even
     to_pad = TRUE;
   }

   const int leftX = (MAXCOLSCR / 2) - (winwidth / 2);
   int rightX = (MAXCOLSCR / 2) + (winwidth / 2);

   if (!to_pad) rightX++;

   const topY = 13;
   const lowY = 15;

   char *s_buf;
   s_buf = new char[ (rightX - leftX + 1 ) * ( lowY - topY + 1 ) * 2 1;
   char *s_buf2;  // right shadow
   s_buf2 = new char[ (1) * ( lowY - topY + 1 ) * 2 I;
   char *s_buf3;  // bottom shadow
   s_buf3 = new char[ (rightX - leftX + 1 ) • (1) • 2 ];

   if (!s_buf II !s_buf2 II !s_buf3) { // failed to allocate memory
     cerr << "\nVideo memory allocation error while prompting\n";
     exit (1);
   }

   gettext(leftX, topY, rightX, lowY, s_buf);
   gettext(rightX + 1, topY + 1, rightX + 1, lowY + 1, s_buf2);
   gettext(leftX + 1, lowY + 1, rightX + 1, lowY + 1, s_buf3);
```

```
    draw-border (leftX, topY, rightX, lowY, BLUE, WHITE);
    gotoxy(leftX + 2, topY + 1); // print the message
    cprintf("%s", msg);
    // paint black spaces
    paint-box(rightX - len - 1, topY + 1, rightX - 2, topY + 1, BLACK);

    // right and bottom shadow
    paint-box(rightX + 1, topY + 1, rightX + 1, lowY + 1, BLACK);
    paint-box(leftX + 1, lowY + 1, rightX + 1, lowY + 1, BLACK);

    use_text_color(WHITE, BLACK); // take the string here
    get_streeng(str_toget, len, rightX - len - 1, topY + 1);
    use_normal_color();

    // restore the previous screen contents, delete the buffer that holds it
    puttext(leftX, topY, rightX, lowY, s_buf);
    delete s_buf;
    puttext(rightX + 1, topY + 1, rightX + 1, lowY + 1, s_buf2);
    delete s_buf2;
    puttext(leftX + 1, lowY + 1, rightX + 1, lowY + 1, s_buf3);
    delete s_buf3;

} // end function ************************** prompt line


// print mssages in the status line
void statusline (int n) {

    char *line1, *line2;
    switch (n) {
      case 1:
        line1 = "Choosing from pull down menu';
        line2 = 'Down, Up, Right, Left, ENTER, ESC";
       break;
      case 2:
        line1 = 'Browsing the SCORES of the student';
        line2 = "Down, Up, PgDown, PgUp, Ctrl-End, Ctrl-Home, ESC";
       break;
      case 3:
        line1 = 'Browsing the results of ITEM ANALYSIS";
        line2 = 'Down, Up, PgDown, PgUp, Ctrl-End, Ctrl-Home, ESC";
       break;
      case 4:
        line1 = "Typing in the filename";
        line2 = "All valid characters in a filename, followed by ENTER";
       break;
      case 5:
        line1 = "Typing in the CLASS name';
        line2 = "Any available character on the keyboard, followed by ENTER";:
       break;
      case 6:
        line1 = 'Typing in the TEST name";
        line2 = 'Any available character on the keyboard, followed by ENTER';
       break;
      case 7:'
        line1 = "Looking at current settings";
        line2 = "Any key (to clear the display)";
       break;
      case 8:
        line1 = 'Choosing the number of choices in a question';
        line2 = 'Down, Up, Right, Left, ENTER, ESC";
       break;
    ) // end switch
```

```
  show-status(line1, line2);

} // end function ************************** statusline


// print out messages on-2 status lines
void show_status (char *line1, char ● lin&Z) {

  // first-line -- current status
  int y = MAXROWSCR - 1;
  int bgcol = RED;
  paint-box- (1, y, MAXCOLSCR, y, bgcol);
  use-text-color(LIGHTGREEN,     bgcol);
  gotoxy(3, y); cprintf("%s", "        [Status:]');
  use-text-color(YELLOW,    bgcol);
  gotoxy(21, y); cprintf("%s", linel);

  // available keys line
  y = MAXROWSCR;
  bgcol = CYAN;
  paint-box (1, y, MAXCOLSCR, y, bgcol);
  use_text_color(RED, bgcol);
  gotoxy(3, y); cprintf("%s", "[Keys available:]");
  use-text-color(BLUE,    bgcol);
  gotoxy(21, y); cprintf("%s", line2);

  use_normal_color();

} // end function * ************************ show_status


// display the class and the test name
void displayclass (char ● claaname, char ● temtname) {

  int y = TITLE-Y + 1;
  int bgcol = CYAN;
  paint-box (1, y, MAXCOLSCR, y, bgcol);
  use-text-color(BLACK,    bgcol);
  gotoxy(lO, y); cprintf("%s", "Class:");
  gotoxy(50, y); cprintf("%s", 'Test:');

  use-text-color(BLUE,    bgcol);
  gotoxy(16, y); cprintf("%s", clasname);
  gotoxy(55, y); cprintf("%s", testname);

} // end function ************************* displayclass


// round a number into 2 decimal places float
float round-to2 (float toround) {

  float x = toround * 100;
  int n = (toround * 100 );

  if (toround > 0) {
    if ( x - n > 0.5) n++;
  } else
    if ( x - n < -0.5) n--;

  return (0.00 + n) / 100;

} // end function ************************* round_to2
```

```
// to_get -- string to be input, lento_get -- len of string to be input
// startX -- X coord to begin getting the string
// Y -- Y coord to get the string
void get-atreeng (char *to-get, int lento_get, int startX, int y) {

  int done = 0, elem = 0;
  char ch;
  gotoxy(startX, y);
  do {
    ch = getche(); // to display while input

    if (ch == ENTER) { // *1*
      if (elem < lento_get - 1) to_get[elem] = '\0'; // null
      done = 1;
    } // *1*
    else
      if (ch == BACKSPACE) ( // ● 2*
       if (elem > 0) {

        if (elem == lento_get - 1) { //     special  case
          gotoxy(startX  +  elem,  y);  // -- because did not erase
          cprintf("%c", ' ');         //     the  last  (max)  char
          gotoxy(startX + elem - 1, y);
        }
        to_get[elem] = '\0';
        cprintf("%c", ' ');
        elem--;
       }
       gotoxy(startX + elem, y);
      } // *2*
      else
        if (ch == ESC) { // *3*
          for (int j = 0; j < lento_get - 1; j++)
            to_get[0] = '\0';
          done = 1;
        } // ● 4*
        else { // ● 5*
          to_get[elem] = ch;
          if (elem < lentoget - 1) elem++;
          gotoxy(startX + elem, y);

        } //  ● 5* end if - else

  } while (!done);

} // end function • *******+**************** get_streeng


// show all the curent settings
void show_settings (int choices, char ● clasname, char *testname,
3  int no-q, int no_students) {

  const int.leftX = 20;
  int rightX = 60;
  const topY = 7;
  const lowY = 17;

  char ● scrn-buf;
  scrn-buf  = new char[ (rightX - leftX + 1 )  * ( lowY - topY + 1 ) * 2 ];
  gettext(leftX, topY, rightX, lowY, scrn-buf);

  // for shadow
  char *scrn_buf2;  // right shadow
  scrn_buf2 = new char[ (1) * ( lowY - topY + 1 ) * 2 ];
  gettext(rightX + 1, topY + 1, rightX + 1, lowY + 1, scrn_buf2);

  char *scrn_buf3;  // bottom shadow
  scrn_buf3 = new char[ (rightX - leftX + 1 ) * (1) * 2 ];
  gettext(leftX + 1, lowY + 1, rightX + 1, lowY + 1, scrn_buf3);

  paint_box(rightX + 1, topY + 1, rightX + 1, lowY + 1, BLACK);
  paint_box(leftX + 1, lowY + 1, rightX + 1, lowY + 1, BLACK);

  draw-border (leftX, topY, rightX, lowY, BLACK, WRITE);

  paint_box(leftX + 1, topY + 1, rightX - 1, lowY - 1, WHITE);
  window (leftX, topY, rightX, lowY);

  use-text-color(BLUE,    WRITE);
  gotoxy(13, 1); cprintf("%s", "µCURRENT SETTINGSÆ");
  gotoxy(3,  2); cprintf("%s", "Score using guessing correction ?:");
  gotoxy(19, 3); cprintf("%s", 'Number of choices:');
  gotoxy(7, 4);  cprintf("%s", 'Display response count in % ?:");

  use-text-color(RED,    WRITE);
  gotoxy(37, 2);
  if (fl_usegc) cprintf("%s", 'YES"); else cprintf("%s", "NO");
  gotoxy(37, 3); cprintf("%d", choices);
  gotoxy(37, 4);
  if (fl_count_perc) cprintf("%s", 'YES'); else cprintf("%s", "NO');

  use-text-color(BLUE,    WRITE);
  gotoxy(11, 6); cprintf("%s", . -- Test information -- ● );
  gotoxy(3, 7);  cprintf("%s", "class :");
  gotoxy(4, 8);  cprintf("%s", "Test  :");
  gotoxy(3, 9);  cprintf("%s", "Number of questions :");
  gotoxy(3, 10); cprintf("%s", "Number of students  :" );

  use-text-color(RED,    WRITE);
  gotoxy(11, 7); cprintf("%s", clasname);
  gotoxy(11, 8);  cprintf("%s", testname);
  gotoxy(25, 9); cprintf("%d", no-q);
  gotoxy(25, 10); cprintf("%d", no-students);

  getch();

// restore the previous screen contents, delete the buffer that holds it
  puttext(leftX, topY, rightX, lowY, scrn-buf);
  delete scrn_buf;
  puttext(rightX + 1, topY + 1, rightX + 1, lowY + 1, scrn_buf2);
  delete scrn_buf2;
  puttext(leftX + 1, lowY + 1, rightX + 1, lowY + 1, scrn_buf3);
  delete scrn_buf3;

  window (1, 1, MAXCOLSCR, MAXROWSCR); // set back to normal
  use_normal_color();

} // end function ************************* show_settings


// take choices from pull down menu, showed through streeng
int sub_pulldown (int leftX, int topY, char streeng[][MAX_M_CHARS]) {

  PullDown subgdmenu (leftX, topY, streeng);
  return sub_pdmenu.choose();

} // end function ************************* sub_pulldown
```