**Introduction:**

One of the main aims of the project is to create a Test Module that can be used for any kind of testing, be it random practice or preparing for competitive exams like NEET, JEE… Different people who’ll use the program will have different needs and they will need different question paper patterns/types (for example NEET exam has only Single Correct Questions while JEE has an amalgam of different question types). So, instead of making question paper types as the primary classes, each question type was made as a primary class. The advantage of this is that any question paper pattern can be made by mixing the required amount of questions of each question type.

**Question Types – The Primary Classes:**

Every primary class has its datamembers to be the questions, options (if part of the question format) and the correct answer.

Every question type class has 5 main member functions (all public):

* A *create* function – To create a question.
* A *display* function – To display a question (either when a student attempts a test/when the student is viewing test analysis)
* A *return\_answer* function – This function returns the answer to a question and is useful during evaluation of a question/displaying the answer, as the answer datamember is a private member of the question paper classes.
* An *anscheck* function – This function compares student response to a question with the correct answer and returns the marks obtained by the student in that question.
* An *analysis* function – Displays analysis of a question after testing/when the student wants to view the answers/analysis (question + correct answer + student response + marks obtained in that question).

An interesting feature of the mcq class is the implementation of partial marking. In case of multiple correct questions, if the correct options are ABC and the student’s response is AB, then the student has partially answered the question and thus partial marks are awarded (+1 for every correct option). This is done with the help of a user-defined function findchar(char\*,char) which helps find a character in a string. Also, in the above scenario, if the student enters BAC/bac, he is awarded full marks as his responses are sorted alphabetically (by the function alphasort(char\*)) and converted to uppercase before evaluation.

Different question papers are made with a mix of questions from each question type.

The advantages of making each question type a class is that now each question can be considered as a separate object with its own functionality and entities. This allows tremendous flexibility to create new question paper types as the need arises,hence making overall maintenance of the program significantly easier, with great scope for expansion/ modification, to always keep it up to date with the latest exam patterns. Also, this ensures that all responses and answers are mapped to the individual question itself, hence providing greater data security and readability.

**Question Paper Types/Patterns – The Secondary Classes:**

The datamembers of a QP class are objects of the primary classes (scq, numerical, mcq, matrix) and the name of the question paper/test. Every QP class has 3 main member functions:

* A *create* function – to create a question paper. This function calls the create function of its question objects.
* A r*esult* function – Once a test has been taken by a student, his/her responses are passed to this function where the student’s score is evaluated using the anscheck functions of the primary classes’ objects. This function is called soon after a student finishes a test and this function displays his/her marks obtained in that test & the number of questions correct/wrong.
* A r*eport* function – After taking up a test, a student would want to analyze his/her performance and view the correct answers. The report function displays the question numbers, the correct answers, the responses of the student and the marks obtained in that question for every question in the test. From here, he/she can see which questions he/she made an error in and can view those questions alone later.

Objects of the QP classes are test papers/ question papers.

**Type classes:**

While QP1 class represents question papers, type1 class represents responses to an exam of type QP1. For instance, QP1 class has 20 SCQ and 5 numericals as data members. So, type1 class has 20 character variables and 5 double variables as datamembers. Similarly, classes type2 and type3 are designed to store student responses to exams of type QP2 and QP3. In addition, each type class has qpname as a datamember so that the responses can be mapped to a question paper.

Type classes have a lot of advantages. With the help of these classes, we can map students’ responses, rather than question papers, to students. This not only saves a lot of memory space (type classes are way smaller than the QP classes) but also makes the program more efficient. The exam paper for which the student has entered the responses can be found out using the qpname datamember of the type classes.

Each type class has a constructor that initializes the initial response of a student (before test attempt) to X/-1 depending on what question type it is (as illustrated above).

**Class Student:**

Class student is used to represent the information/details of a student. Objects of this class are the students(test takers) themselves. The data members of this class are name, ID, password, request to admin(a string), standard & section, and multiple objects of class type1, type2, type3. ID and password of a student will serve as the login credentials of a student. Main functions of the student class:

* *Create* function – This function gets the basic details of a newly created student object (a student who has recently created a profile to practice tests using this program) from the user (student).
* *Display* function – This function displays the current details of a student.
* *Edit* function – If a student/admin wants to edit the details of a student, this function is called where the current details are displayed and a field can be edited.
* *Return functions* [ret\_id(), ret\_name(), ret\_pass()] which return private members whenever needed.

**Other classes:**

There are also small classes like institute\_details (data members = number of question papers and names of available question papers) and admin (data members are username and password of admin).

**Files used:**

6 binary files are used:

* Student.dat – To store details of students
* Admin.dat – To store username and password of admin
* Inst.dat – To store names of all question papers and number of question papers
* QP1.dat – To store question papers of type QP1
* QP2.dat – To store question papers of type QP2
* QP3.dat – To store question papers of type QP3

**Main UDFs in the program:**

There are 2 kinds of users to the program: Admin and Student.

1) *Start\_procedure* (void start\_procedure()):

The admin starts tests for students. Start\_procedure is a UDF in the program where the admin enters the ID of student who will take up the test. He/She then enters the QP type (1/2/3) and QP Name of the test to be started for the student. The question paper is ‘downloaded’ from the binary file it is present in and the admin is asked if a student is reattempting a test. If so, all previously entered responses to that question paper are deleted. Then the student logs in using his credentials and starts the exam. Control then goes to the UDF Test\_Procedure (1/2/3) [there are 3 test procedure functions – one for each question paper pattern].

2) *Test\_Procedure1/2/3* (void test\_procedure(student &, qp&, int):

In test\_procedure (1/2/3), the student can view questions and answer them. Instructions are displayed before start of test. The responses of the test taker are also displayed during the test so that the test taker can have a track of their responses. This is done with the help of small UDFs disp\_res(student&, int) (1/2/3) which display the responses of the student. After answering a question, the student can enter N to go to the next question, Q to choose the question he/she wants to go to, I to view instructions again & S to submit. After the student finishes the test, the student submits the paper. The results of his test are immediately computed using the *result* function of the QP class whose object is the testpaper he/she attempted and his score is displayed. The student is then given the option of viewing his test analysis. If student enters Y, control shifts to the test analysis function (1/2/3) [one for each question paper pattern]. Else, test procedure ends.

3) *Test\_analysis1/2/3* (void test\_analysis(student &, qp&, int):

Test analysis (1/2/3), a UDF, creates a comprehensive test report by showing the question using the *report* function of QP class (1/2/3). The student his then shown his mark split-up in each QType and then can view each question and its answer separately. After viewing a question in test analysis, the student can enter N to go to the next question, Q to choose the question he/she wants to go to & X to exit. Once the student is done, he can exit from the test analysis. The test analysis of a test taken by a student can be viewed by the admin too.

**Other UDFs:**

Some small UDFs present in the program help in transfer of data from binary file to console and vice-versa.

* voidupload\_newstu\_details(student&) – uploads new student details onto Student.dat
* voidmodify\_stu\_details(student&) – uploads modified details of existing student onto Student.dat
* void addqp1(qp1&), void addqp2(qp2&), void addqp3(qp3&) – Add new question papers onto the Question\_Paper files
* int finqp1(char\*,qp1&), int findqp2(char\*, qp2&), int findqp3(char\*, qp3&) – Get question papers from QPfiles, – returns 1 if paper is found, 0 if not
* get\_institute() – Get institute details from Inst.dat
* put\_institute() – Put (modified) institute details from program to Inst.dat
* get\_admin() - Get admin username and password from Admin.dat
* put\_admin() - Put (modified) admin details from program to Admin.dat

**ADMIN:**

Admin Home screen:

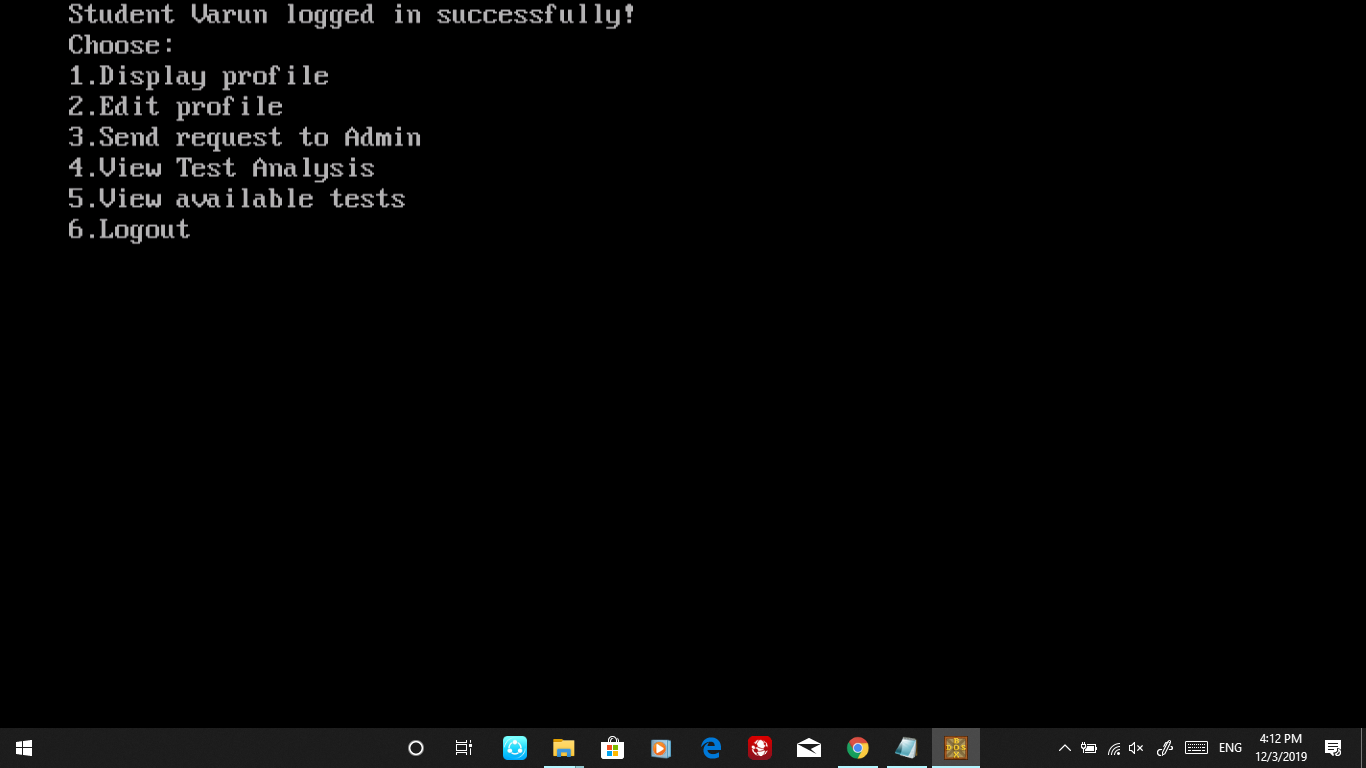


Functionalities available:

1. Creating question papers: The admin is the administrator of the program and he/she makes question papers for exams. If the admin wants to create a question paper of some type say QP2, the *create* function of class QP2 comes into play. The created paper is then stored on its corresponding binary file.
2. Viewing available tests: The admin can view the tests created so far. This feature helps him/her know the tests available. This information is taken from the file Inst.dat.
3. Start Test Procedure:The start\_procedure function is called. A description of this function has been provided before and the flow of control from there is as illustrated:
4. Create new student: The admin can enroll a new student/add a new student’s details to the files here. The *create* function of the class student is invoked and students’ details are inputted. They are then saved onto the file Student.dat.
5. View Student details: In this feature, the admin is shown the list of all students and asks him to choose. He can choose a student and view their profile. This information is extracted from the binary file & then display function of student displays his/her info.
6. Modify student details: In this feature, the admin is shown the list of all students and asks him to choose. He can choose a student and view their profile. This information is extracted from the binary file & then display function of student displays his/her current info. The user can then choose which detail of student to edit. The *edit* function of class student is invoked here. After making changes, the student details are modified on the file.
7. View student requests: The admin can also view any requests sent to him from students. Request is a data member of class student. Once the admin has seen a request, he/she has the option to make the request null or let it stay. This can serve as an indicator to the student to whether the admin has seen the request or not.
8. View student test analysis: The admin can view the test analysis of any test taken by any student. The admin chooses the student from the list of students displayed, enter the test type(1/2/3) and QPName of the test the admin wants to see. The testpaper is extracted from the binary file. Control then flows to the test\_analysis function. Once done, the admin can exit the analysis.
9. Change login details: The admin can change his/her login details (username & password). After change, these are updated on the file Admin.dat.

**STUDENT:**

Student home screen:



Functionalities available:

1. View my details: Once a student logs in, his/her details will be stored on a student object, let’s say stu. The *display* function of this object displays the details/profile of the student.
2. Edit my details: The *edit* function of stu will be called; this displays the current details of a student and the student can choose which detail to edit. After making changes, the student details are modified on the file Student.dat.
3. Enter request to admin: The student can request the admin for a test. His request is stored on a string and displayed to the admin whenever he/she wants to view requests. After the admin views a request, he/she can make it void, indicating that the request has been seen by the admin.
4. View test analysis: The student enters the test type(1/2/3) and QPName of the test he/she wants to see analysis for. The testpaper is extracted from the binary file. Control then flows to the test\_analysis function. Once done, the student can exit the analysis.
5. View available tests: The student can view the tests created so far. This feature helps him/her know the tests available. This information is taken from the file Inst.dat.