Software Design Specifications Document

For

STUDENT REGISTRATION SYSTEM

Submitted To-

Mr.M.R.Warsi

Prepared by –

Omar Raghib (14PEB017)

Rifa Khan (14PEB054)

Anushka Chawla (14PEB073)

Amir Raza (13PEB303)

(Group Number 4)

Dated-

28th March 2016

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this software design document is to provide a low level description of the Student Registration System, providing insight into the structure and design of each component. Topics covered include the following-

- Function declarations and interactions.
- Data flow and design.
- Processing narratives.
- File Handling techniques.
- Algorithmic models.
- Design constraints and restrictions
- User Interface design
- Test cases and expected results.

In short, this document is to equip the reader with a thorough understanding of the inner workings of the Student Registration System.

1.2 Document Conventions

The text below has been typed as per the following scheme-

- All section headings have been typed in Times New Roman font with bold style in size 20.
- All section sub headings have been typed in Calibri font with bold style in size 16.
- The text of the document has been typed in Times New Roman font in size 13.
- The captions of tables and figures have been typed in Times New Roman font with Italics style in size 12.
- The user data in figures has been typed in italics while the screen's data has been typed in normal style.
- The names of the structs and text files used have been capitalized and marked in bold style.

1.3 Intended Audience and Reading Suggestions

This document has been written as such to be of use to the personnel directly involved in the development of the Student Registration System i.e. the members of the development team, the testing team, project consultants and any other technical staff associated with this project.

1.4 Product Scope

This document will be used by the following people-

- Development team to develop a suitable source code for the project.
- The course in charge for assessment of assignment.
- The course in charge for analyzing the suitability of various coding techniques in the code.
- The course in charge for testing of functionalities provided in the code.

1.5 References

The conception of this project was greatly inspired and aided by the following sources-

- "An Integrated Approach to Software Engineering", by Pankaj Jalote.
- https://web.cs.dal.ca/~arc/.../Templates/Design%20Templates/SDS.doc
- https://www.ieee.org/education_careers/education/standards/remote_controlled_dissolved_oxygen_fett_student_application_paper.pdf
- https://en.wikipedia.org/wiki/**Template**:**IEEE_software**_documents

2. System Overview

The Student Registration System will consist of a C program where data about each student will be represented by a struct student and data about each faculty member will be represented by the struct faculty. Data of all students will be stored in a text file and similarly, data of all faculty members will be stored in a separate text file. An additional text file having the enrolment number of all the students will be used to check the validation of user id entered during login process.

The functionalities of the software will be achieved by functions which will operate on the data of the above mentioned files.

The enrolment number of all the students along with their passwords will be mentioned in a separate file so that checking the validity of an enrolment number during login becomes possible. Each student will use his enrolment number as his user id. During the login process, the system will check the entered user id against the list of enrolment numbers written in the file. If a match is found, it will proceed to check the password of the student. The password will also be stored in the file containing records of students and it will be user defined in case of students. Pre-defined passwords would be used in the case of faculty members which would be written on a separate text file.
The system features can be broken down into two groups:

(i) Core features which are essential to the function of the application (registration of a student, viewing a student's profile)
(ii) Additional features which are only meant to add extra functionalities (number of students in a particular year/department/course).

Every time an error occurs, an error message is displayed, offering options to either go back to the main menu or exit the program.
The registration system is equipped to count the total number of credits in a semester. If the total number of credits is greater than 30, an error message is flashed.

After the successful execution of a program, a logout option appears for exiting the program.

3. System Architecture

3.1 Architectural Design

The software structure consists of two structs-

1. struct **STUDENT**-to store the details about students like their name, phone number, enrolment number, password etc.
2. struct **FACULTY**-to store the details about faculty members like their name and password.

The data of the software would be stored in four files-

1. A text file **ENROL** storing the enrolment numbers and the passwords of all the students against which the valid enrolment number will be checked at the time of registration and login. At the time of logging in, both the enrolment numbers and passwords will be checked.
2. A text file **STUDENT** containing the details about all the registered students.
3. A text file **FACULTY** containing the names and passwords of all the faculty members.
4. A text file **SUGGEST** containing the suggestions of users.

The functions designed to display and modify data actually work upon the data of instances of the structs and the structs will receive their data from the files.
Initially **ENROL** contains the enrolment number and a fixed password for all the students. This password is supposed to be changed by the student while doing registration. The details of all the students (including the courses they have registered for) is stored in **STUDENT**. Whenever a student enters his login id (which is his enrolment number) for registration, it is compared with the enrolment number of all the students in **ENROL**-

- In case a match is found, the software prompts the user to enter his details and set a password.
- In case no match is found, the software gives an error message and offers options to the user to either go back to the home screen or exit the program.

The next time when a student uses the software, he enters both- his login id (his enrolment number) and the password. Like before, the enrolment number will be compared with the enrolment number of all the students in **ENROL** -

- In case a match is found, the software prompts the user to enter his password. It then checks the password in a similar fashion-
    - In case a match is found, it redirects the user to a menu of functions
    - In case no match is found, it redirects the user back to login screen from where he can either enter new details or exit the program.
- In case no match is found, the software gives an error message and offers options to the user to either go back to the home screen or exit the program.

A similar login procedure is followed for faculty members with the only difference that they don't have to set their passwords…they would be given pre-defined passwords which would be stored in **FACULTY**.

Once logged in, students can register themselves for various courses of their semester by choosing the register function from the list of functions displayed. During registration, the logical significance of various entries (like the consistency of the branch with the entered faculty number, etc.) is checked with the help of various conditions applied in the code. In case of any discrepancy, an error message is displayed. All the data entered during registration is first stored in an instance of struct **STUDENT** which is then as a whole written on the file **STUDENT**.

After registration, the student can view all of his entered details using the studentProfile function which when called picks up the required data from **STUDENT** file and prints it on the console screen. This function also offers the facility to modify certain details like phone number, password etc.

Both students and faculty members can view the names and total number of students registered in a particular course/year/department by entering the required course number/year/department(branch) while the miscellaneous users can view only the total number of students in any of these cases. This is done by passing the required course number/year/department's name as an argument to the corresponding function. This function then compares the argument passed with the details in **STUDENT** file and displays the corresponding details if a match is found.

The suggestions function is available to all kinds of users-students, faculty members and miscellaneous users. It asks the user to enter a suggestion which is then written on **SUGGEST** file.

3.3 Design Rationale

- Files have been used to store data because the data has to be stored permanently.
- Text files are used to store passwords because the passwords are allotted by the development team.
- Struct is used because a structure contains a number of data types grouped together and hence all the details of a student can be stored in a single structure.

4. Data Design

4.1 Data Description

The program uses global variables like choice, counter etc. to accept user input and keep track of the number of credits. It uses two structs- struct **FACULTY** and struct **STUDENT** whose fields store the data.

Struct **STUDENT** consists of the following fields-

1. enno
2. facno
3. firstname
4. midname
5. lastname
6. branch
7. year
8. semester
9. phone
10. email
11. backlog
12. courses
13. backlog_courses

Struct **FACULTY** consists of the following fields-

1. name
2. pwd.

All necessary variables along with the fields of the struct are listed in the data dictionary included below. The programmer may take more variables apart from these to satisfy the needs of the code.

4.2 Data Dictionary

Variable Name	DataType	Description
enno	String	Contains the enrolment number of the student.
facno	String	Contains the faculty number of the student.
firstname	String	Contains the first name of the student.
midname	String	Contains the middle name of the student.
lastname	String	Contains the last name of the student.
branch	String	Contains the branch of the student.
Year	Integer	Contains the year in which the student is enrolled at present.
semester	Integer	Contains the semester in which the student is enrolled at present.
phone	String	Contains the phone number of the student.
email	String	Contains the email id of the student.
backlog	Character	Contains 'Y' if a student has a backlog and contains 'N' if he doesn't.
courses	string array	Contains the list of the courses in which a student is registered.
backlog_courses	string array	Contains the list of the courses in which a student has backlog (if any).
name	String	Contains the name of faculty members.
Pwd	String	Contains the password of the faculty members.
credits	Integer	Contains the total number of credits applied for by the student in the entered semester.
counter	Integer	Counts the total number of students whose records satisfy a given condition.
choice	Integer	Accepts the user's choice in response to the menu asking if he is a student/faculty

		member/miscellaneous user.
userid	String	Accepts the enrolment number of the student as a string to use it as his userid during login.

5. Function Design-

5.1-) for students-

5.1.1-) login Function

int login(int choice)	
Input	An integer 'choice' to decide if the login has to be implemented for a student or a faculty member
Output	Returns 1 if an account exists for the entered login id and password else returns 0.
Description	The function prompts the user to enter a login id and password. The variable 'choice' entered determines the file against which the entered login id and password have to be checked (**ENROL** in case of students and **FACULTY** in case of faculty members). In case a match is found, the function returns 1 else it returns 0.

5.1.2-) register Function

void register()	
Input	None
Output	None
Description	The function prompts the user to enter enno ,facno, firstname, midname, lastname, branch, year, semester, phone, email, backlog, courses, backlog_courses, credits etc. The logical significance of the data is checked with respect to various conditions. In case of any discrepancy an error message is displayed. All this data is first stored in an instance of struct **STUDENT** and this instance as a whole is then written onto **STUDENT** file.

5.1.3-) student profile Function

void studentProfile()	
Input	None
Output	The record of the student is displayed.
Description	The function displays the record of the logged in student and provides options to the user to modify his details. This is done by checking the enrolment number of all the students whose detail is stored in file **STUDENT**. In case a match is found, the corresponding data is picked up in an instance of struct **STUDENT** and the contents of that instance are then displayed on the console screen.

5.1.4-) coursewise_students Function

void coursewise_students(char courseno[])	
Input	A string ‘courseno’ to determine the course number names of whose students have to be displayed.
Output	The names of the students enrolled in that course are displayed. The total number of the students enrolled in that course is also displayed.
Description	The function accepts a string ‘courseno’ as an argument and then compares that string against the records of courses of all the registered students stored in file **STUDENT**. In case it finds a match of the entered course number in the record of a student, it picks up the detail of the student in an instance of struct **STUDENT** and displays the name of that particular student on console screen. This matching procedure is carried out for all student records. Simultaneously, a counter counts the total number of matches found. At the end of the search operation, this number is displayed as the total number of students enrolled in that course.

5.1.5-) yearwise_students Function

void yearwise_students(int year)	
Input	An integer ‘year’ to determine the year names of students enrolled in which have to be displayed.
Output	The names of the students enrolled in that year are displayed. The total number of the students enrolled in that year is also displayed.
Description	The function accepts an integer ‘year’ as an argument and then compares that integer against the records of courses of all the registered students stored in file **STUDENT**. In case it finds a match of the entered year in the record of a student, it picks up the detail of the student in an instance of struct **STUDENT** and displays the name of that particular student on console screen. This matching procedure is carried out for all student records. Simultaneously, a counter counts the total number of matches found. At the end of the search operation, this number is displayed as the total number of students enrolled in that year.

5.1.6-) departmentwise_students Function

void departmentwise_students(char dept[])	
Input	A string ‘dept’ to determine the department names of students enrolled in which have to be displayed.
Output	The names of the students enrolled in that department are displayed. The total number of the students enrolled in that department is also displayed.
Description	The function accepts a string ‘dept’ as an argument and then compares that string against the records of courses of all the registered students stored in file **STUDENT**. In case it finds a match of the entered department (branch) in the record of a student, it picks up the detail of the student in an instance of struct **STUDENT** and displays the name of that particular student on console screen. This matching procedure is carried out for all student records. Simultaneously, a counter counts the total number of matches found. At the end of the search operation, this number is displayed as the total number of students enrolled in that department.

5.1.7-) all_backlog_details Function

void all_backlog_details()	
Input	None
Output	The names of the students with backlog(s) are displayed along with the course(s) in which they have backlog. The total number of the students having backlog is also displayed.
Description	The function checks the backlog status of each registered student in the records tored in file **STUDENT**. In case it finds it to be 'Y' in a record it displays the name of that particular student along with the course in which he has backlog. This matching procedure is carried out for all student records. Simultaneously, a counter counts the total number of students having a backlog. This total number is displayed after all the records have been scanned.

5.1.8-) coursewise_backlog Function

void coursewise_backlog(char courseno[])	
Input	A string 'courseno' to determine the course number names of students having backlog in which have to be displayed.
Output	The names of the students having backlog in that course are displayed. The total number of the students having backlog in that course is also displayed.
Description	The function accepts a string 'courseno' from the user and then compares that string against the records of backlog courses of all the registered students stored in file **STUDENT**. In case it finds a match of the entered course number in the record of a student, it displays the name of that particular student. This matching procedure is carried out for all student records. Simultaneously, a counter counts the total number of students having a backlog. This total number is displayed after all the records have been scanned.

5.1.9-) suggestions Function

void suggestions()	
Input	None
Output	None
Description	The function accepts suggestions from the user and writes them on a file SUGGEST.

5.2-) For Faculty Members-

5.2.1-) login Function

int login(int choice)	
Input	An integer ‘choice’ to decide if the login has to be implemented for a student or a faculty member
Output	Returns 1 if an account exists for the entered login id and password else returns 0.
Description	The function prompts the user to enter a login id and password. The variable ‘choice’ entered determines the file against which the entered login id and password have to be checked (**ENROL** in case of students and **FACULTY** in case of faculty members). In case a match is found, the function returns 1 else it returns 0.

5.2.2-) coursewise_students Function

void coursewise_students(char courseno[])	
Input	A string 'courseno' to determine the course number names of whose students have to be displayed.
Output	The names of the students enrolled in that course are displayed. The total number of the students enrolled in that course is also displayed.
Description	The function accepts a string 'courseno' as an argument and then compares that string against the records of courses of all the registered students stored in file **STUDENT**. In case it finds a match of the entered course number in the record of a student, it picks up the detail of the student in an instance of struct **STUDENT** and displays the name of that particular student on console screen. This matching procedure is carried out for all student records. Simultaneously, a counter counts the total number of matches found. At the end of the search operation, this number is displayed as the total number of students enrolled in that course.

5.2.3-) yearwise_students Function

void yearwise_students(int year)	
Input	An integer ‘year’ to determine the year names of students enrolled in which have to be displayed.
Output	The names of the students enrolled in that year are displayed. The total number of the students enrolled in that year is also displayed.
Description	The function accepts an integer ‘year’ as an argument and then compares that integer against the records of courses of all the registered students stored in file **STUDENT**. In case it finds a match of the entered year in the record of a student, it picks up the detail of the student in an instance of struct **STUDENT** and displays the name of that particular student on console screen. This matching procedure is carried out for all student records. Simultaneously, a counter counts the total number of matches found. At the end of the search operation, this number is displayed as the total number of students enrolled in that year.

5.2.4-) departmentwise_students Function

void departmentwise_students(char dept[])	
Input	A string ‘dept’ to determine the department names of students enrolled in which have to be displayed.
Output	The names of the students enrolled in that department are displayed. The total number of the students enrolled in that department is also displayed.
Description	The function accepts a string ‘dept’ as an argument and then compares that string against the records of courses of all the registered students stored in file **STUDENT**. In case it finds a match of the entered department (branch) in the record of a student, it picks up the detail of the student in an instance of struct **STUDENT** and displays the name of that particular student on console screen. This matching procedure is carried out for all student records. Simultaneously, a counter counts the total number of matches found. At the end of the search operation, this number is displayed as the total number of students enrolled in that department.

5.2.5-) all_backlog_details Function

void all_backlog_details()	
Input	None
Output	The names of the students with backlog(s) are displayed along with the course(s) in which they have backlog. The total number of the students having backlog is also displayed.
Description	The function checks the backlog status of each registered student in the records tored in file **STUDENT**. In case it finds it to be 'Y' in a record it displays the name of that particular student along with the course in which he has backlog. This matching procedure is carried out for all student records. Simultaneously, a counter counts the total number of students having a backlog. This total number is displayed after all the records have been scanned.

5.2.6-) coursewise_backlog Function

void coursewise_backlog(char courseno[])	
Input	A string 'courseno' to determine the course number names of students having backlog in which have to be displayed.
Output	The names of the students having backlog in that course are displayed. The total number of the students having backlog in that course is also displayed.
Description	The function accepts a string 'courseno' from the user and then compares that string against the records of backlog courses of all the registered students stored in file STUDENT. In case it finds a match of the entered course number in the record of a student, it displays the name of that particular student. This matching procedure is carried out for all student records. Simultaneously, a counter counts the total number of students having a backlog. This total number is displayed after all the records have been scanned.

5.2.7-) suggestions Function

void suggestions()	
Input	None
Output	None
Description	The function accepts suggestions from the user and writes them on a file SUGGEST.

5.3-) For Miscellaneous Users-

5.3.1-) coursewise_students Function

void coursewise_students(char courseno[])	
Input	A string 'courseno' to determine the course number names of whose students have to be displayed.
Output	The total number of the students enrolled in that course is displayed.
Description	The function accepts a string 'courseno' as an argument and then compares that string against the records of courses of all the registered students stored in file **STUDENT**. In case it finds a match in the record it increments the counter by 1.After scanning all the records, it displays the current value of counter as the total number of students enrolled in that course.

5.3.2-) yearwise_students Function

void yearwise_students(int year)	
Input	An integer ‘year’ to determine the year names of students enrolled in which have to be displayed.
Output	The total number of the students enrolled in that year is displayed.
Description	The function accepts an integer ‘year’ as an argument and then compares that string against the records of all the registered students stored in file **STUDENT**. In case it finds a match in the record it increments the counter by 1.After scanning all the records, it displays the current value of counter as the total number of students enrolled in that year.

5.3.3-) departmentwise_students Function

void departmentwise_students(char dept[])	
Input	A string ‘dept’ to determine the department names of students enrolled in which have to be displayed.
Output	The total number of the students enrolled in that department is displayed.
Description	The function accepts a string ‘dept’ as an argument and then compares that string against the records of department of all the registered students stored in file **STUDENT**. In case it finds a match in the record it increments the counter by 1.After scanning all the records, it displays the current value of counter as the total number of students enrolled in that department.

5.3.4-) all_backlog_details Function

void all_backlog_details()	
Input	None
Output	The total number of the students having backlog(s) is displayed.
Description	The function checks the backlog status of each registered student. In case it finds it to be 'Y' in a record it increments the counter by 1.After scanning all the records, it displays the current value of counter as the total number of students having backlog(s).

5.3.5-) coursewise_backlog Function

void coursewise_backlog(char courseno[])	
Input	A string 'courseno' to determine the course number names of students having backlog in which have to be displayed.
Output	The total number of the students having backlog in that course is displayed.
Description	The function accepts a string 'courseno' as an argument and then compares that string against the records of backlog courses of all the registered students stored n file **STUDENT**. In case it finds a match in the records it increments the counter by 1.After scanning all the records, it displays the current value of counter as the total number of students having backlog in that course.

5.3.6-) suggestions Function

void suggestions()	
Input	None
Output	None
Description	The function accepts suggestions from the user and writes them on a file SUGGEST.

7. Pseudo code

7.1-) Login function-

```
 //choice is accepted as an argument for the function
 CHECK choice
 IF choice is equal to 1
{
 ask the user to INPUT login id and password.
 OPEN the file ENROL.
 COMPARE login id and corresponding password with each entry of the file.
 IF a match is found
 STORE userid in a global variable
 RETURN 1
 ELSE
 RETURN 0
 CLOSE the file.
}
ELSE IF choice is equal to 2
{
 ask the user to INPUT login id and password.
 OPEN the file FACULTY.
 COMPARE login id and corresponding password with each entry of the file.
 IF a match is found
 RETURN 1
 ELSE
 RETURN 0
 CLOSE the file.
}
ENDIF
```

7.2-)Register Function-

```
 Open STUDENT file.
 INITIALIZE credits equal to 0.
 CHECK the enrolment number in STUDENT file
 IF a record corresponding to the enrolment number already exists
   DISPLAY error message
 ELSE
{
  ask the user to INPUT new password
  OPEN ENROL file
  UPDATE password corresponding to the enrolment number entered by user in ENROL file.
  Ask the user to INPUT his/her firstname, midname and lastname.     //In case user wants to skip a
particular field he/she must enter a '-'.
 Ask the user to choose from the displayed list of branches.
 IF choice of branch is not equal to 1 or 2
  DISPLAY error message.
 ELSE
```

```
  The branch corresponding to the choice of branch is checked against the faculty number.
 IF it is consistent with the faculty number
  Ask the user to INPUT the year of engineering
IF the entered year is less than 1 or greater than 4
  DISPLAY error message
ELSE
 The year of engineering is saved.
Ask the user to choose his semester from the displayed list
IF the entered choice is consistent with the list
 SAVE the semester
ELSE
 DISPLAY error message
Ask the user to INPUT phone and email.
DISPLAY the list of courses according to the year entered.
ENTER choice
IF the course entered is a theory course
 Add 4 to the credits.
ELSE IF the course entered is a laboratory course
 Add 2 to the credits.
ENDIF
Ask the user to INPUT backlog status
IF the backlog status is Y
{
  DISPLAY the relevant list of courses
  Ask user to ENTER choice
  IF the course entered is a theory course
   Add 4 to the credits.
ELSE IF the course entered is a laboratory course
  Add 2 to the credits.
ENDIF
}
IF credits is greater than 30
 DISPLAY error message
ENDIF
Ask the user if he wants to submit
IF the user enters Y
{
 WRITE the data of struct student on the STUDENT file.
}
ELSE
 DISPLAY navigation menu
CLOSE STUDENT file
CLOSE ENROL file.
```

7.3-)Student Profile function-

```
 DISPLAY a menu to choose from modifying or just displaying a record
 Ask the user to enter his choice
IF choice is equal to 1
{
  OPEN STUDENT file.
 CHECK all the records until the end of file is encountered.
 IF the entered enrolment number is not found
{
  DISPLAY error message
}
 ELSE
{
  READ the record corresponding to the enrolment number.
 STORE the record in an instance of struct STUDENT.
 DISPLAY the data of the struct STUDENT.
}
}
 ELSE IF choice is equal to 2
{
  OPEN STUDENT file.
  CHECK all the records until the end of file is encountered.
 IF the entered enrolment number is not found
{
 DISPLAY error message
}
ELSE
{
 DISPLAY a menu asking the user the field in which he wishes to make the changes-password,
phone number or email id.
 Ask the user to ENTER his choice.
 IF choice is less than 1 or greater than 3
{
  DISPLAY error message.
}
 ELSE
{
 Ask the user to ENTER the new data of the selected field.
 MODIFY the record in STUDENT file
 Ask the user if he wants to make more changes
 IF the user ENTERS y,
{
 CALL the studentProfile function.
}
}
}
```

7.4-)coursewise_students function

```
 OPEN STUDENT file.
 INITIALIZE Counter equal to 0.
 WHILE end of file is not reached
{
 TRAVERSE each record.
 COMPARE each record for the course number entered.
 IF a match is found
{
 INCREMENT counter.
 IF choice is equal to 1 or 2
{
 PRINT the name of the student.
}
}
}
PRINT Counter.
CLOSE STUDENT file.
```

7.5-)yearwise_students function

```
 OPEN STUDENT file.
 INITIALIZE Counter equal to 0.
 WHILE end of file is not reached
{
 TRAVERSE each record.
 COMPARE each record for the year entered.
 IF a match is found
{
 INCREMENT counter.
 IF choice is equal to 1 or 2
{
 PRINT the name of the student.
}
}
}
 PRINT Counter.
CLOSE STUDENT file.
```

7.6-)departmentwise_students function

```
 OPEN STUDENT file.
 INITIALIZE Counter equal to 0.
 WHILE end of file is not reached
{
 TRAVERSE each record.
 COMPARE each record for the branch entered.
 IF a match is found
{
 INCREMENT counter.
 IF choice is equal to 1 or 2
{
 PRINT the name of the student.
}
}
}
 PRINT Counter.
 CLOSE STUDENT file
```

7.7-)all_backlog_details function

```
OPEN STUDENT file.
INITIALIZE Counter equal to 0.
WHILE end of file is not reached
{
 TRAVERSE each record.
 CHECK each record for the backlog status.
 IF backlog status is Y
{
 INCREMENT counter.
 IF choice is equal to 1 or 2
{
 PRINT the name of the student and the corresponding course(s) in which he has the backlog.
}
}
}
PRINT Counter.
CLOSE STUDENT file.
```

7.8-)coursewise_backlog function

```
OPEN STUDENT file.
INITIALIZE Counter equal to 0.
WHILE end of file is not reached
{
 TRAVERSE each record
 CHECK each record for the backlog status.
 IF backlog status is equal to Y
{
  COMPARE the entered course number with the course number of  the backlog course.
  IF a match is found
{
   INCREMENT counter.
   IF choice is equal to 1 or 2
{
   PRINT the name of the student.
}
}
}
}
 PRINT Counter.
 CLOSE STUDENT file
```

7.9-)suggestions function-

```
OPEN SUGGEST file.
Ask the user to ENTER a suggestion.
WRITE the suggestion on the SUGGEST file
CLOSE SUGGEST file.
```

*****End of the SDS*****