
MARIANO MARCOS STATE UNIVERSITY
DEPARTMENT OF COMPUTING AND INFORMATION SCIENCES
BACHELOR OF SCIENCE IN INFORMATION TECHNOLOGY

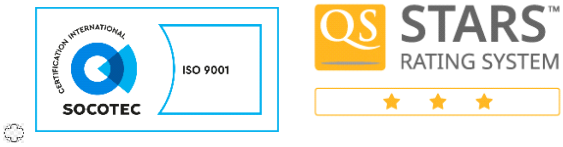
# Student Record Management System

## (Inventory Management System)

SUBMITTED BY:
Labuguen, Marc Luis B.

SUBMITTED TO:
Manuel, Harbie Jay L.

SYSTEM THESIS DOCUMENTATION
MAY 2023




## INTRODUCTION

The Student Record Management System is a comprehensive software program developed in C++ to manage and manipulate student records efficiently. It serves as a centralized platform for educational institutions, such as schools and colleges, to store, organize, and process student information effectively. The system provides various functionalities, including adding, displaying, searching, modifying, and deleting student records, allowing administrators and faculty members to streamline their record-keeping tasks.

In today's digital age, educational institutions face the challenge of managing vast amounts of student data. Traditional paper-based record-keeping methods are cumbersome, time-consuming, and prone to errors. The Student Record Management System addresses these challenges by providing an automated solution to handle student records electronically. By digitizing the record management process, educational institutions can save valuable time and resources while ensuring accuracy and data integrity.

## OBJECTIVE

The objective of the Student Record Management System is to provide educational institutions with an efficient and reliable software solution for managing student records. The system aims to streamline the record-keeping process, eliminate manual paperwork, and ensure accurate and accessible student information. By leveraging the power of C++ programming, the system offers a user-friendly interface and a wide range of functionalities to facilitate the management, organization, and manipulation of student records. Its key objectives include:

- Centralized Record Management: The system provides a centralized platform for storing, organizing, and managing student records, allowing educational institutions to access and update information easily.
- Efficient Data Handling: By automating record management tasks, the system minimizes errors and saves valuable time and resources. It offers functions for adding, displaying, searching, modifying, and deleting student records, enabling efficient data handling.
- Data Integrity and Security: The system ensures the integrity and security of student data. It incorporates file handling mechanisms to save and retrieve records securely, preventing unauthorized access and data loss.
- User-Friendly Interface: The system features a user-friendly interface, making it easy for administrators and faculty members to navigate and perform various operations on student records. It provides clear menu options and intuitive functionalities.
- Sorting and Searching Capabilities: The system allows sorting student records based on roll numbers, facilitating quick and easy data retrieval. It also offers a search function to find specific records based on roll numbers.
- File Persistence: The system supports saving student records to a file, enabling long-term data persistence. It allows educational institutions to retrieve records from a file, ensuring data continuity and historical reference.

- Scalability and Adaptability: The system is designed to accommodate the evolving needs of educational institutions. It can handle a large number of student records and can be easily modified and extended to incorporate additional features as required.

By achieving these objectives, the Student Record Management System aims to enhance the efficiency, accuracy, and accessibility of student record management processes within educational institutions, ultimately improving administrative productivity and facilitating effective decision-making.

## COMPONENTS OF STUDENT RECORD MANAGEMENT SYSTEM

- **Database Management:**

  The Database Management component is responsible for storing and retrieving inventory data efficiently. It utilizes a database management system (DBMS) to create, manage, and manipulate the inventory database. The component includes tables, relationships, and queries that enable the storage and retrieval of information such as item details, stock levels, pricing, suppliers, and transaction history. By effectively managing the inventory database, this component ensures data integrity, accuracy, and consistency throughout the system.

- **Inventory Tracking and Control:**

  The Inventory Tracking and Control component play a vital role in monitoring and managing inventory levels in real-time. It enables businesses to track the movement of items, monitor stock levels, and implement automatic reorder points. This component facilitates inventory control mechanisms such as barcode scanning, RFID technology, or manual entry to update inventory records, record sales and purchases, and trigger alerts when stock levels fall below a certain threshold. By providing accurate and up-to-date information, this component helps businesses avoid stockouts, minimize excess inventory, and improve overall inventory management efficiency..

- **Inventory Tracking and Control:**
- The Inventory Tracking and Control component play a vital role in monitoring and managing inventory levels in real-time. It enables businesses to track the movement of items, monitor stock levels, and implement automatic reorder points. This component facilitates inventory control mechanisms such as barcode scanning, RFID technology, or manual entry to update inventory records, record sales and purchases, and trigger alerts when stock levels fall below a certain threshold. By providing accurate and up-to-date information, this component helps businesses avoid stockouts, minimize excess inventory, and improve overall inventory management efficiency.
- **Reporting and Analytics:**

The Reporting and Analytics component enables businesses to generate comprehensive reports and perform data analysis to gain insights into inventory performance and trends. It includes features for generating inventory reports such as stock levels, sales analysis, purchase history, and profitability. Additionally, this component may incorporate data visualization tools to present the information in a visually appealing and easily understandable format, allowing businesses to make informed decisions regarding inventory planning, purchasing, and sales strategies.

## REFERENCES


Admin, "Download Dev-C++ 5.11 for Windows 10, 7, 8 (32-64 bit)," *FileHippo - Download Free Software Latest 2022*, 23-Feb-2019. [Online]. Available: https://www.fileshipposoftwares.com/dev-c-download-for-windows-10/#System_Requirements_For_DEV-C_For_Window_10.

: https://www.stroustrup.com/the_c++_programming_language.html

GeeksforGeeks: https://www.geeksforgeeks.org/

Stack Overflow: https://stackoverflow.com/

## CODE DOCUMENTATION

```
#include <iostream>
#include <fstream>
#include <conio.h>
#include <string.h>
#include <cstdlib>

#define MAX 20

```

- The iostream library is included to provide input and output functionalities in C++. It allows you to perform standard input and output operations, such as reading from and writing to the console.
- The conio.h library is a header file specific to the Microsoft Visual C++ compiler. It provides functions for console input and output operations. In this code, it is used to call the getch() function, which waits for a keypress from the user before continuing execution. This is commonly used to pause the program and wait for user input.
- The fstream library is included to handle file input and output operations. It provides classes and functions that allow you to read from and write to files. In this code, it is likely used to perform operations on files, such as reading data from a file or writing data to a file.
- The string.h library is included to provide functions for manipulating strings in C++. It contains functions for operations such as string concatenation, comparison, copying, and more. In this code, it might be used for handling string-related operations, such as manipulating student names or other textual data.
- The cstdlib library provides functions for general-purpose operations in C++. It includes functions such as memory allocation, converting strings to numeric values, generating random numbers, and more. In this code, it is likely included for some

specific functionality, such as generating random values or performing other general-purpose operations.

```
11  typedef struct date
12  {
13      int dd,mm,yyyy,choice;
14  }DATE;
```

This code defines a structure called d**ate** using the **typedef** keyword. The structure has four integer members: **dd** (day), **mm** (month)**, yyyy** (year), and choice. This structure is used to represent a date with day, month, and year values.

```
16  typedef struct STUD
17  {
18      int StudentNo;
19      char firstName[10];
20      char lastName[10];
21      int marks[3];
22      float average;
23      DATE d;
24  }STUD;
25
```

This code defines another structure called STUD, which represents a student record. It has the following members:

- **StudentNo** of type **int**: Represents the student number.
- **firstName** of type **char[10]:** Represents the first name of the student.
- **lastName** of type **char[10]**: Represents the last name of the student.
- **marks** of type **int[3]:** Represents an array of three integers to store the marks of the student.
- **average** of type **float**: Represents the average marks of the student.
- **d** of type **DATE**: Represents a date structure (as defined earlier) to store the date of birth or some other relevant date for the student.

```
26  int num;
```

This declares an integer variable num without initializing it. It is likely used to store the number of students or some other count used in the program.

```
28  // Function prototypes
29  int Menu();
30  int SortMenu();
31  void Login();
32
33  void AddStudent(STUD [MAX],int);
34  void DisplayRecords(STUD [MAX]);
35
36  void Sort(STUD [MAX]);
37  void SortByFirstName(STUD [MAX]);
38  void SortByLastName(STUD [MAX]);
39  void SortByStudentNo(STUD [MAX]);
40  void SortByAverage(STUD [MAX]);
41  void SortByDOB(STUD [MAX]);
42
43  void Search(STUD [MAX]);
44  void Modify(STUD [MAX]);
45  void Delete(STUD [MAX]);
46
47  void SaveToFile(STUD [MAX]);
48  void RetriveData(STUD [MAX]);
```

These lines are function prototypes. They declare the names and signatures of various functions used in the code, allowing the compiler to recognize them before they are defined. This allows the functions to be called from other parts of the code. The function prototypes listed include **Menu(), SortMenu(), Login(), AddStudent(), DisplayRecords(), Sort(), SortByFirstName(), SortByLastName(), SortByStudentNo(), SortByAverage(), SortByDOB(), Search(), Modify(), Delete(), SaveToFile(), and RetrieveData()**. The **STUD[MAX]** parameter indicates that these functions take an array of **STUD** structures as an argument.

These function prototypes provide an outline of the functions used in the code and help with code organization and readability. The actual implementation of these functions can be found later in the code or in separate files.

```
50  int main()
51  {
52  STUD s[MAX];
53  int choice,no;
54  RetriveData(s);
55  Login();
56  while(1)
57  {
58      choice=Menu();
59      switch(choice)
60      {
61          case 1:
62              cout << "\n\t\t\t\t\t\t\t\t\t\t\t   Enter the no of Students:- ";
63              cin >> no;
64              AddStudent(s,no);
65              break;
66
67          case 2:
68              DisplayRecords(s);
69              break;
70
71          case 3:
72              Sort(s);
73              break;
74
75          case 4:
76              Search(s);
77              break;
78
79          case 5:
80              Modify(s);
81              break;
82
83          case 6:
84              Delete(s);
85              break;
86
87          case 7:
88              SaveToFile(s);
89              break;
90
91          case 0:
92              exit(0);
93
94          default:
95              cout << "\n\t\t\t\t\t\t\t\t\t\t\t  Enter correct Choice\n";
96              getch();
97              system("cls");
98              break;
99      }
100   }
101 }
102
```

The **main()** function is the entry point of the program and is where the execution of the program begins.

1. Declare variables:
    - **STUD s[MAX]**: Declares an array of **STUD** structures named s. This array will store student records.

- **int choice, no**: Declares two integer variables **choice** and **no. choice** will store the user's menu choice, and no will store the number of students entered by the user.

2.Retrieve data:

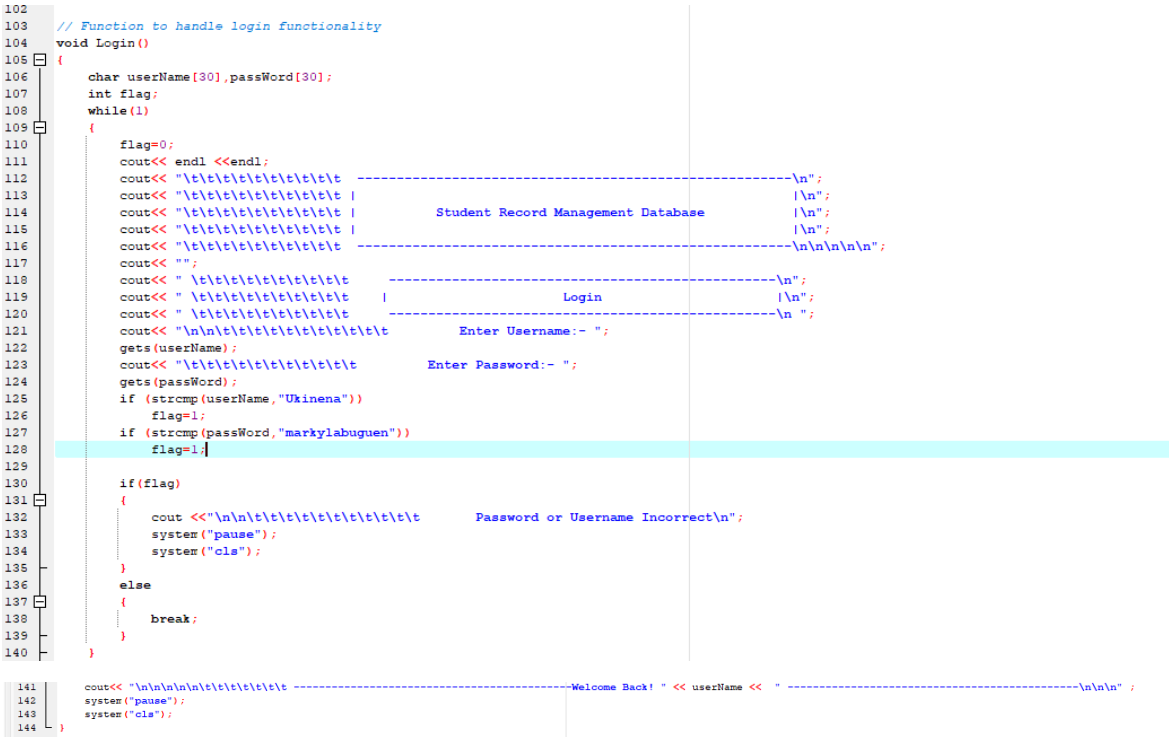
- **RetrieveData(s):** Calls **the RetrieveData()** function to load previously saved student records from a file into the **s** array. This ensures that any previously entered data is available for manipulation.

3..Login:

- **Login():** Calls the **Login()** function, which likely handles user authentication or authorization before granting access to the program's functionality. This ensures that only authorized users can use the system.

4.Menu-driven loop:

- **while (1):** Starts an infinite loop to display the menu and handle user choices until the program is manually exited.
- **choice = Menu():** Calls the **Menu()** function to display the menu options and retrieve the user's choice. The value returned by **Menu()** is assigned to the choice variable.
- **switch (choice):** Based on the value of choice, the program enters a specific case block to perform the corresponding action.
- Case 1: Add Student
- Prompts the user to enter the number of students.
- Calls the **AddStudent()** function, passing the **s** array and the number of students as arguments. This function allows the user to enter the details of the specified number of students and adds them to the s array.
- Case 2: Display Records
- Calls the **DisplayRecords()** function, passing the **s** array as an argument. This function displays the records of all students stored in the **s** array.
- Case 3: Sort Records
- Calls **the Sort() function**, passing the **s** array as an argument. This function provides options to sort the student records based on different criteria like first name, last name, student number, average marks, or date of birth.
- Case 4: Search Records
- Calls the **Search()** function, passing the **s** array as an argument. This function allows the user to search for a specific student record based on criteria like student number, first name, or last name



```
// Function to handle login functionality
void Login()
{
    char userName[30],passWord[30];
    int flag;
    while(1)
    {
        flag=0;
        cout<< endl <<endl;
        cout<< "\t\t\t\t\t\t\t\t\t\t\t  ------------------------------------------------------\n";
        cout<< "\t\t\t\t\t\t\t\t\t\t\t |                                                    |\n";
        cout<< "\t\t\t\t\t\t\t\t\t\t\t |          Student Record Management Database        |\n";
        cout<< "\t\t\t\t\t\t\t\t\t\t\t |                                                    |\n";
        cout<< "\t\t\t\t\t\t\t\t\t\t\t  ------------------------------------------------------\n\n\n\n\n";
        cout<< "";
        cout<< " \t\t\t\t\t\t\t\t\t\t\t     ------------------------------------------------------\n";
        cout<< " \t\t\t\t\t\t\t\t\t\t\t    |                      Login                       |\n";
        cout<< " \t\t\t\t\t\t\t\t\t\t\t     ------------------------------------------------------\n ";
        cout<< "\n\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t         Enter Username:- ";
        gets(userName);
        cout<< "\t\t\t\t\t\t\t\t\t\t\t\t         Enter Password:- ";
        gets(passWord);
        if (strcmp(userName,"Ukinena"))
            flag=1;
        if (strcmp(passWord,"markylabuguen"))
            flag=1;

        if(flag)
        {
            cout <<"\n\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t       Password or Username Incorrect\n";
            system("pause");
            system("cls");
        }
        else
        {
            break;
        }
    }
    cout<< "\n\n\n\n\n\n\t\t\t\t\t\t\t ----------------------------------------Welcome Back! " << userName <<  " ----------------------------------------\n\n\n" ;
    system("pause");
    system("cls");
}
```

The **Login()** function handles the login functionality of the program.

1. Declare variables:

   - **char userName[30], passWord[30]:** Declare two character arrays **userName** and **passWord** to store the entered username and password, respectively.
   - **int flag**: Declare an integer variable **flag** to indicate whether the entered username or password is incorrect.

2. Login loop:

   - The function enters an infinite loop using **while (1)** to repeatedly prompt the user for their username and password until the correct credentials are entered or the user manually exits the program.

3. Display login screen:

   - The function displays a login screen with a program title and login form using various **cout** statements. This provides a visually pleasing and informative interface for the user.

4. Retrieve username and password:

   - The function prompts the user to enter their username and password using **gets()** function calls. The entered values are stored in the **userName** and **passWord** variables, respectively.

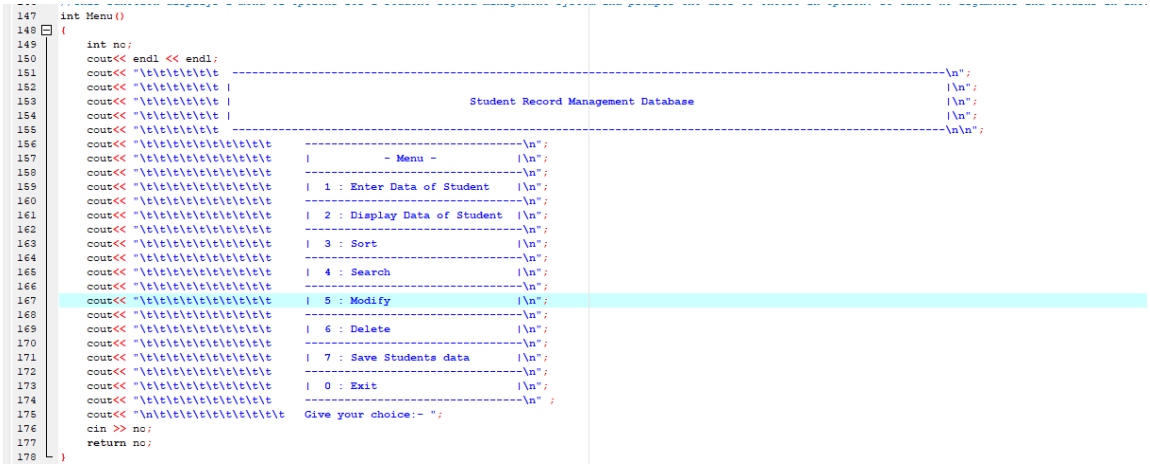
5. Check credentials:

- The function compares the entered username with the string "Ukinena" using **strcmp()** function. If the comparison result is non-zero (indicating inequality), the **flag** variable is set to 1, indicating that the username is incorrect.
- Similarly, the function compares the entered password with the string "markylabuguen". If the comparison result is non-zero, the **flag** variable is set to 1, indicating that the password is incorrect.

6. Handle incorrect credentials:

- If the **flag** variable is non-zero (indicating incorrect username or password.

---



```
int Menu()
{
    int no;
    cout<< endl << endl;
    cout<< "\t\t\t\t\t\t  ------------------------------------------------------------------------------------------------------------\n";
    cout<< "\t\t\t\t\t\t |                                                                                                          |\n";
    cout<< "\t\t\t\t\t\t |                                  Student Record Management Database                                      |\n";
    cout<< "\t\t\t\t\t\t |                                                                                                          |\n";
    cout<< "\t\t\t\t\t\t  ------------------------------------------------------------------------------------------------------------\n\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     |           - Menu -             |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     |  1 : Enter Data of Student     |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     |  2 : Display Data of Student   |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     |  3 : Sort                      |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     |  4 : Search                    |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     |  5 : Modify                    |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     |  6 : Delete                    |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     |  7 : Save Students data        |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     |  0 : Exit                      |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t     ----------------------------------\n" ;
    cout<< "\n\t\t\t\t\t\t\t\t\t\t\t   Give your choice:- ";
    cin >> no;
    return no;
}
```

The **Menu()** function displays a menu of options for the student record management system and prompts the user to choose an option.

1.Declare variable:

- **int no**: Declare an integer variable no to store the user's choice.

2.Display menu:

- The function uses multiple **cout** statements to display the menu interface. Each **cout** statement displays a specific option or title of the menu.
- The menu includes options such as entering data of a student, displaying student data, sorting records, searching records, modifying records, deleting records, saving student data, and exiting the program.

3.Get user's choice:

- The function prompts the user to enter their choice using **cin >> no**. The entered value is stored in the **no** variable.

4.Return user's choice:

- The function returns the value of **no**, which represents the user's choice.

This function is used in the main program to determine the user's selected option and perform the corresponding actions based on that choice.

```
int SortMenu()
{
    int no;
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    |          - Sort Menu -         |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    |  1 : Sort by First Name        |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    |  2 : Sort by Last Name         |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    |  3 : Sort by Student No        |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    |  4 : Sort by Average Marks     |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    |  5 : Sort by Date Of Birth     |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    ----------------------------------\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    |  0 : Exit from Sort Menu       |\n";
    cout<< "\t\t\t\t\t\t\t\t\t\t\t    ----------------------------------\n";
    cout<< "\n\t\t\t\t\t\t\t\t\t\t\t   Give your choice:- ";
    cin >> no;
    return no;
}
```

The SortMenu() function displays a menu of sorting options for the student records and prompts the user to choose a sorting option.

1. Declare variable:

   - **int no:** Declare an integer variable **no** to store the user's choice.

2. Display menu:

   - The function uses multiple **cout** statements to display the sorting menu interface. Each **cout** statement displays a specific sorting option.
   - The menu includes options such as sorting by first name, last name, student number, average marks, and date of birth.

3. Get user's choice:

   - The function prompts the user to enter their choice using **cin >> no**. The entered value is stored in the no variable.

4. Return user's choice:

   - The function returns the value of no, which represents the user's choice.

This function is used in the program when the user chooses the "Sort" option from the main menu. It allows the user to select the sorting criteria for the student records.

---

```
//This function allows the user to add student records to the database. It takes an array of STUD structures s and the number of students no to add.
//It prompts the user to enter details for each student (first name, last name, student number, marks in math, science, and English, and date of birth) and stores them in the s array.//
void AddStudent(STUD s[MAX],int no)
{
    for(int i=num;i<num+no;i++)
    {
        cin.ignore();
        cout << "\n\t\t\t\t\t\t\t\t\t\t\t   Enter details for Student " << i+1 << endl;
        cout << "\t\t\t\t\t\t\t\t\t\t\t   First Name:- ";
        cin.getline(s[i].firstName, MAX);
        cout << "\t\t\t\t\t\t\t\t\t\t\t   Last Name:- ";
        cin.getline(s[i].lastName, MAX);
        cout << "\t\t\t\t\t\t\t\t\t\t\t   Student No:(8- digits):- ";
        cin >> s[i].StudentNo;
        cout << "\t\t\t\t\t\t\t\t\t\t\t   Marks obtained in Maths, Science and English (Each out of 100):- ";
        cin >> s[i].marks[0] >> s[i].marks[1] >> s[i].marks[2];
        s[i].average = (float)(s[i].marks[0]+s[i].marks[1]+s[i].marks[2])/3;
        cout << "\t\t\t\t\t\t\t\t\t\t\t   Enter Date, Month and Year of Birth:- ";
        cin >> s[i].d.dd >> s[i].d.mm >> s[i].d.yyyy;
    }
    num=num+no;
    system("cls");
}
```

The AddStudent() function allows the user to add student records to the database

1. Parameters:

- **STUD s[MAX]:** An array of structures of type **STUD** representing the student records.
- **int no**: An integer representing the number of students to add.

2.Loop:

- The function uses a **for** loop to iterate over the range of num **to num + no - 1,** where **num** is a global variable that keeps track of the number of students currently stored.

This loop allows the user to enter details for each student.

3.Prompting for student details:

- The function prompts the user to enter details for each student, including their first name, last name, student number, marks in math, science, and English, and date of birth.
- **cin.getline()** is used to read the first name and last name, allowing spaces to be included in the input.
- The student's marks are stored in the **marks** array, and the average mark is calculated by summing the marks and dividing by 3.

5. Updating the student count:

- After adding the new students, the **num** variable is updated by incrementing it by **no**.
- This ensures that the next set of students will be added at the correct position in the array.

6.Clearing the screen:

- The function uses **system("cls")** to clear the screen after adding the students.

This function is used in the program when the user chooses the "Enter Data of Student" option from the main menu. It allows the user to add multiple student records to the database.

---

```
230  void DisplayRecords(STUD s[MAX])
231  {
232      int i;
233      cout << "\n\t\t\t\t\t  ------------------------------------------------------------------------------------------------------------------------";
234      cout << "\n\t\t\t\t\t  | Record No. | Student No. | Full Name           | Math Marks | Science Marks | English Marks | Average Marks | Date of Birth |";
235      cout << "\n\t\t\t\t\t  ------------------------------------------------------------------------------------------------------------------------";
236      for(i=0;i<num;i++)
237      {
238          string fullName = std::string(s[i].firstName) + " " + s[i].lastName;
239          printf("\n\t\t\t\t\t  | %10d | %11d | %-20s | %10d | %13d | %13d | %13.2f | %02d/%02d/%04d |",
240        i+1, s[i].StudentNo, fullName.c_str(),
241        s[i].marks[0], s[i].marks[1], s[i].marks[2], s[i].average,
242        s[i].d.dd, s[i].d.mm, s[i].d.yyyy);
243
244      }
245      cout << "\n\t\t\t\t\t  ------------------------------------------------------------------------------------------------------------------------\n";
246      cin.get();
247      system("pause");
248      system("cls");
249  }
```

The **DisplayRecords()** function is responsible for displaying the student records stored in the STUD array.

1.Parameter:

- **STUD s[MAX]:** An array of structures of type **STUD** representing the student records.

2.Displaying the table header:

- The function uses cout statements to print the table header, which includes the column names for various student details.

3.Loop:

- The function uses a **for** loop to iterate over the range of **0 to num - 1,** where num is a global variable representing the number of students currently stored.
- This loop iterates through each student record and displays their details in the table format.

4. Formatting and printing student records:

- Inside the loop, the function creates a string **fullName** by concatenating the first name and last name of the student with a space in between.
- The student details are printed using the **printf** function, which allows for formatted output.

Each student's record is printed on a separate line, with appropriate formatting for each field.

5.Displaying the table footer:

- After printing all the student records, the function prints a table footer to visually separate the records.

6. Clearing the screen:

- The function uses **cin.get(), system("pause"),** and **system("cls")** to wait for user input and clear the screen after displaying the records.

This function is used in the program when the user selects the "Display Data of Student" option from the main menu. It prints the student records in a tabular format for easy viewing.

---

```
252  void swap(STUD *x,STUD *y)
253  {
254      STUD temp = *x;
255      *x = *y;
256      *y = temp;
257  }
```

The swap() function is a utility function used in sorting algorithms to swap the values of two STUD structures.

1. Parameters:
   - **STUD *x:** A pointer to a **STUD** structure.
   - **STUD *y**: A pointer to another **STUD** structure.
2. Swapping the values:
   - The function uses a temporary variable of type **STUD** called **temp**.

- The value pointed to by **x** is assigned to **temp**, effectively creating a copy of the **STUD** structure pointed to by **x**.
- The value pointed to by **y** is assigned to the memory location pointed to by **x**, replacing the value originally stored there.
- Finally, the value stored in temp (which is the original value of **x**) is assigned to the memory location pointed to by **y**, completing the swap operation.

This function allows for the swapping of **STUD** structures, which is useful in sorting algorithms when rearranging elements based on specific criteria. By swapping the values of two structures, you can change their order in an array or other data structure.

The **swap()** function is typically used in sorting algorithms like bubble sort, selection sort, or insertion sort to rearrange elements based on a specific sorting criterion, such as student names or grades.

---

```
261  void Sort(STUD s[MAX])
262  {
263      int choice;
264      char choice2;
265      system("cls");
266      while(1)
267      {
268          cout<<endl<<endl;
269       cout << "\t\t\t\t\t\t\t\t    ------------------------------------------------------------------------------------\n";
270      cout << "\t\t\t\t\t\t\t\t    |                                                                                  |\n";
271      cout << "\t\t\t\t\t\t\t\t    |                        Student Record Management Database                        |\n";
272      cout << "\t\t\t\t\t\t\t\t    |                                                                                  |\n";
273       cout << "\t\t\t\t\t\t\t\t    ------------------------------------------------------------------------------------\n\n";
274          choice=SortMenu();
275          switch(choice)
276          {
277              case 1:
278                  SortByFirstName(s);
279                  break;
280
281              case 2:
282                  SortByLastName(s);
283                  break;
284
285              case 3:
286                  SortByStudentNo(s);
287                  break;
288
289              case 4:
290                  SortByAverage(s);
291                  break;
292
293              case 5:
294                  SortByDOB(s);
295                  break;
296
297              case 0:
298                  system("cls");
299                  return;
300
301              default:
302                  std::cout << "\n\t\t\t\t\t\t\t\t\t\t\t\t   Enter correct Choice\n";
303                  break;
304          }
305          cin.ignore();
306          cout << "\n\t\t\t\t\t\t\t\t\t\t\t   Do you want to store sorted records? - ";
307          cin >> choice2;
308          if(choice2 =='y' || choice2 == 'Y')
309              SaveToFile(s);
310          else
311              system("cls");
312      }
313  }
```

The **Sort()** function is responsible for displaying the student record management database menu, handling user input, and calling the appropriate sorting functions based on the user's choice.

1. Initialization:

   - The function initializes two variables: **choice** to store the user's choice from the menu, and **choice2** to store the user's decision on whether to store the sorted records.
   - It also clears the console screen using the **system("cls")** command.

2. Menu and user input:

   - The function enters a while loop that continues indefinitely until the user chooses to exit.
   - Inside the loop, it displays the student record management database menu and prompts the user to make a choice.
   - The **SortMenu()** function is called to display the menu and retrieve the user's choice, which is stored in the **choice** variable.

3. Sorting and handling user input:

   - Based on the user's choice, a corresponding case in the switch statement is executed.
   - Each case calls a specific sorting **function (SortByFirstName(), SortByLastName(), etc.)** to sort the student records accordingly.
   - After sorting, the program prompts the user whether they want to store the sorted records. The user's input is stored in the choice2 variable.

4. Storing sorted records:

   - if the user chooses to store the sorted records**(choice2 == 'y' || choice2== 'Y'),** the **SaveToFile()** function is called to save the records to a file.
   - If the user chooses not to store the records, the console screen is cleared using **system("cls").**

5. Loop continuation or exit:

   - The loop continues to the next iteration, displaying the menu again, allowing the user to chose another sorting option or exit the program by selecting 0.
   - If the user selects 0, the program exits the **Sort()** function and returns to the calling code.

This function provides a menu-driven interface for sorting student records and gives the user the option to store the sorted records for future use.

---

```
315 //This function sorts the student records based on their first names in ascending order. It takes an array of STUD structures s as an argument and modifies the array accordingly.
316 void SortByFirstName(STUD s[MAX])
317 {
318     int i, j;
319     char choice;
320     for (i = 0; i < num-1; i++)
321     {
322         for (j = 0; j < num-i-1; j++)
323         {
324             if (strcmp(s[j].firstName,s[j+1].firstName)>0)
325               swap(&s[j],&s[j+1]);
326         }
327     }
328     cout <<"\n\t\t\t\t\t\t\t\t\t\t\t\t   Records Sorted\n";
329 }
```

The **SortByFirstName()** function is responsible for sorting the student records based on their first names in ascending order.

1. Initialization:
   - The function declares three variables: **i** and **j** for loop counters, and choice to store user input (not used in this function).
   - The loop counters are initialized to 0.

2. Sorting:
   - The function uses nested loops to compare and swap the student records based on their first names.
   - The outer loop **(i)** iterates from 0 to num-1, where num represents the total number of student records.
   - The inner loop **(j)** iterates from 0 to num-i-1, comparing adjacent student records.
   - Inside the loop, the strcmp() function is used to compare the first names of two student records **(s[j].firstName and s[j+1].firstName).**
   - If the comparison result is greater than 0, it means that the first name of **s[j]** is lexicographically greater than the first name of **s[j+1],** indicating that they are out of order.
   - In that case, the swap() function is called to swap the positions of s[j] and s[j+1] in the array.

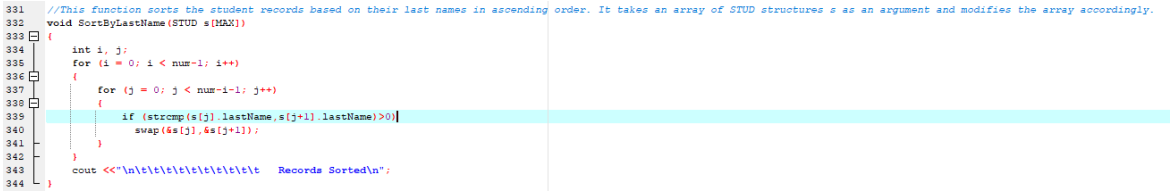
3.Displaying the result:
   - After the sorting process is complete, the function outputs the message "\n\t\t\t\t\t\t\t\t\t\t\t Records Sorted" to indicate that the records have been sorted successfully.

The sorting algorithm used in this function is Bubble Sort, which compares adjacent elements and swaps them if they are in the wrong order. The process is repeated until the entire array is sorted in ascending order based on the first names.

It's important to note that the **swap()** function, which is used within **SortByFirstName(),** is a utility function responsible for swapping the values of two STUD structures (student records) by taking their memory addresses as input.

Overall, **SortByFirstName()** provides the functionality to sort the student records based on their first names, allowing for better organization and retrieval of data.

---



```
331  //This function sorts the student records based on their last names in ascending order. It takes an array of STUD structures s as an argument and modifies the array accordingly.
332  void SortByLastName(STUD s[MAX])
333  {
334      int i, j;
335      for (i = 0; i < num-1; i++)
336      {
337          for (j = 0; j < num-i-1; j++)
338          {
339              if (strcmp(s[j].lastName,s[j+1].lastName)>0)
340                  swap(&s[j],&s[j+1]);
341          }
342      }
343      cout <<"\n\t\t\t\t\t\t\t\t\t\t\t   Records Sorted\n";
344  }
```

The **SortByLastName()** function is responsible for sorting the student records based on their last names in ascending order.

1. Initialization:
   - The function declares two variables**, i** and **j,** as loop counters.
2. Sorting:
   - The function uses nested loops to compare and swap the student records based on their last names.
   - The outer loop **(i)** iterates from 0 to **num-1**, where **num** represents the total number of student records.
   - The inner loop **(j)** iterates from 0 to **num-i-1,** comparing adjacent student records.
   - Inside the loop, the strcmp() function is used to compare the last names of two student records **(s[j].lastName and s[j+1].lastName).**
   - If the comparison result is greater than 0, it means that the last name of s[j] is lexicographically greater than the last name of **s[j+1],** indicating that they are out of order.
   - In that case, the swap() function is called to swap the positions of s[j] and **s[j+1]** in the array.
3. Displaying the result:
   - After the sorting process is complete, the function outputs the message "\n\t\t\t\t\t\t\t\t\t\t Records Sorted" to indicate that the records have been sorted successfully.

The sorting algorithm used in this function is Bubble Sort, which compares adjacent elements and swaps them if they are in the wrong order. The process is repeated until the entire array is sorted in ascending order based on the last names.

It's important to note that the **swap()** function, which is used within **SortByLastName(),** is a utility function responsible for swapping the values of two **STUD** structures (student records) by taking their memory addresses as input.

Overall, **SortByLastName()** provides the functionality to sort the student records based on their last names, allowing for better organization and retrieval of data.

---

```
346 //This function sorts the student records based on their student numbers in ascending order. It takes an array of STUD structures s as an argument and modifies the array accordingly.
347 void SortByStudentNo(STUD s[MAX])
348 {
349     int i, j;
350     for (i = 0; i < num-1; i++)
351     {
352         for (j = 0; j < num-i-1; j++)
353         {
354             if (s[j].StudentNo > s[j+1].StudentNo)
355               swap(&s[j],&s[j+1]);
356         }
357     }
358     cout <<"\n\t\t\t\t\t\t\t\t\t\t\t   Records Sorted\n";
359 }
```

The **SortByStudentNo()** function is responsible for sorting the student records based on their student numbers in ascending order. It uses a simple bubble sort algorithm to compare each pair of adjacent records and swap them if they are in the wrong order.

The outer loop iterates from the first element to the second-to-last element of the array. The inner loop compares and swaps elements, moving from the first element to the last unsorted element in each iteration of the outer loop. This process is repeated until the entire array is sorted.

Inside the inner loop, the function compares the **StudentNo** field of the current element s[j] with the **StudentNo** field of the next element **s[j+1].** If the StudentNo of the current element is greater than the StudentNo of the next element, it indicates that the records are out of order, and a swap is performed using the **swap()** function.

After sorting the records, the function outputs a message indicating that the records have been sorted.

It's worth noting that the **num** variable used in the loops is not defined in the provided code snippet. You'll need to ensure that **num** is correctly defined and represents the number of elements in the array **s** before calling this function.

---

```
363  void SortByAverage(STUD s[MAX])
364  {
365      int i,j,pos;
366      STUD temp;
367      for (i = 0; i < num-1; i++)
368      {
369          pos=i;
370          for (j = i+1;j<num;j++)
371          {
372              if (s[j].average < s[pos].average)
373                pos=j;
374          }
375          if(pos!=i)
376          {
377              swap(&s[i],&s[pos]);
378          }
379      }
380      cout <<"\n\t\t\t\t\t\t\t\t\t\t\t   Records Sorted\n";
381  }
```

The **SortByAverage()** function is responsible for sorting the student records based on their average marks in ascending order. It uses a selection sort algorithm to find the smallest average mark in each iteration and swaps it with the current element being considered.

The outer loop iterates from the first element to the second-to-last element of the array. In each iteration, it assumes that the current element at **s[i]** has the smallest average mark and initializes the pos variable with i.

The inner loop then starts from **i+1** and compares the average mark of each subsequent element with the average mark of the assumed smallest element **(s[pos]).** If a smaller average mark is found, the **pos** variable is updated to store the index of that element.

After iterating through the inner loop, the function checks if the **pos** variable has changed from its initial value of **i.** If it has, it means that a smaller average mark was found, and a swap is performed between the element at **s[i]** and the element at **s[pos]** using the **swap()** function.

This process continues until the entire array is sorted in ascending order based on the average marks.

Finally, the function outputs a message indicating that the records have been sorted.

It's worth noting that the **num** variable used in the loops is not defined in the provided code snippet. You'll need to ensure that **num** is correctly defined and represents the number of elements in the array **s** before calling this function.

---

```
384  void SortByDOB(STUD s[MAX])
385  {
386      int i,j;
387      for (i=0;i<num-1;i++)
388      {
389          for(j=0;j<num-1-i;j++)
390          {
391              if(s[j].d.yyyy>s[j+1].d.yyyy)
392              {
393                  swap(&s[j],&s[j+1]);
394              }
395              if(s[j].d.yyyy == s[j+1].d.yyyy && s[j].d.mm>s[j+1].d.mm)
396              {
397                  swap(&s[j],&s[j+1]);
398              }
399              if(s[j].d.yyyy == s[j+1].d.yyyy && s[j].d.mm == s[j+1].d.mm && s[j].d.dd>s[j+1].d.dd)
400              {
401                  swap(&s[j],&s[j+1]);
402              }
403          }
404      }
405      cout <<"\n\t\t\t\t\t\t\t\t\t\t\t   Records Sorted\n";
406  }
```

The **SortByDOB()** function is responsible for sorting the student records based on their date of birth (DOB) in ascending order. It uses a bubble sort algorithm to compare and swap elements based on their DOB components (year, month, and day).

```
410  void Search(STUD s[MAX])
411  {
412      int no,j,flag=0;
413      std::cout << "\n\t\t\t\t\t\t\t\t\t\t\t   Enter the Student No of the student to search:- ";
414      std::cin >> no;
415      for(j=0;j<num;j++)
416      {
417          if(s[j].StudentNo==no)
418          {
419              cout << "\n\t\t\t\t\t\t\t\t\t\t\t   Record Found...\n";
420              cout << "\n\t\t\t\t\t\t\t\t\t\t\t   First Name:- " << s[j].firstName;
421              cout << "\n\t\t\t\t\t\t\t\t\t\t\t   Last Name:- " << s[j].lastName;
422              cout << "\n\t\t\t\t\t\t\t\t\t\t\t   Student No:- " << s[j].StudentNo;
423              cout << "\n\t\t\t\t\t\t\t\t\t\t\t   Marks obtained in Maths, Science and English (Each out of 100):- " << s[j].marks[0] << " " << s[j].marks[1] << " " << s[j].marks[2];
424              cout << "\n\t\t\t\t\t\t\t\t\t\t\t   Average Marks Obtained:- " << s[j].average;
425              cout << "\n\t\t\t\t\t\t\t\t\t\t\t   Date, Month and Year of Birth:- " << s[j].d.dd << "/" << s[j].d.mm << "/" << s[j].d.yyyy << endl;
426              flag++;
427              cin.get();
428              system("pause");
429              system("cls");
430              break;
431          }
432      }
```

```
433      if(!flag)
434      {
435          cout << "\n\t\t\t\t\t\t\t\t\t\t\t   Record with Student No " << no << " not found..." << endl;
436          cin.get();
437          system("pause");
438          system("cls");
439      }
440  }
```

The **Search()** function is used to search for a student record based on their student number **(StudentNo).** It prompts the user to enter the student number they want to search for.

The function then iterates through the array of **STUD** structures using a **for** loop, checking if the S**tudentNo** of each element matches the inputted student number.

If a match is found **(s[j].StudentNo == no),** it means the record has been found. The function prints the details of the student, such as first name, last name, student number, marks obtained in different subjects, average marks, and date of birth.

---

```
444  void Modify(STUD s[MAX])
445  {
446      int no,j,flag=0;
447      cout << "\n\t\t\t\t\t\t\t\t\t\t\t   Enter the Student No of the student to modify:- ";
448      cin >> no;
449      for(j=0;j<num;j++)
450      {
451          if(s[j].StudentNo==no)
452          {
453              flag=1;
454              break;
455          }
456      }
```

1. function **Modify()** takes an array of **STUD** structures as a parameter.

2. it prompts the user to enter the student number they want to modify.

3. it declares variables **no** to store the inputted student number, j as a loop counter, and **flag** to indicate whether a matching student record is found (initially set to 0).

4. The function then enters a **for** loop that iterates over the num number of elements in the s array. The variable **num** is not defined within the provided code snippet, so its value is unclear. Please ensure num is defined appropriately before using it in the loop.

5. Within the loop, it checks if the **StudentNo** of the current element **(s[j].StudentNo)** is equal to the inputted student number **(no).** If a match is found, it sets the **flag** variable to 1, indicating that a matching student record has been found, and breaks out of the loop.

---

```
488  void Delete(STUD s[MAX])
489  {
490      int j,no,flag=0;
491      cout  <<"\n\t\t\t\t\t\t\t\t\t\t\t   Enter the Student No of the student to Delete:- ";
492      cin >>  no;
493      for (j = 0; j < num; j++)
494      {
495          if (no == s[j].StudentNo)
496          {
497              flag++;
498              for (int k = j; k < num-1; k++)
499                  s[k] = s[k + 1];
500              num--;
501              cout <<"\n\t\t\t\t\t\t\t\t\t\t\t   Record deleted successfully\n";
502              break;
503          }
504      }
505      if(flag)
506          SaveToFile(s);
507      else
508      {
509          cout <<"\n\t\t\t\t\t\t\t\t\t\t\t   Record with Student No"<< no <<" not found...";
510          getch();
511          system("pause");
512          system("cls");
513      }
514  }
```

1. The function prompts the user to enter the student number of the student to be deleted.

2. A loop is used to search for a matching student number in the **s** array. If a match is found, the code inside the if block is executed.

3. Inside the **if** block, the **flag** variable is incremented to indicate that a matching record was found.

4. A nested loop is used to shift the elements of the s array to the left, starting from the index j, effectively deleting the record at index j.

5. The **num** variable is decremented to reflect the reduced number of records.

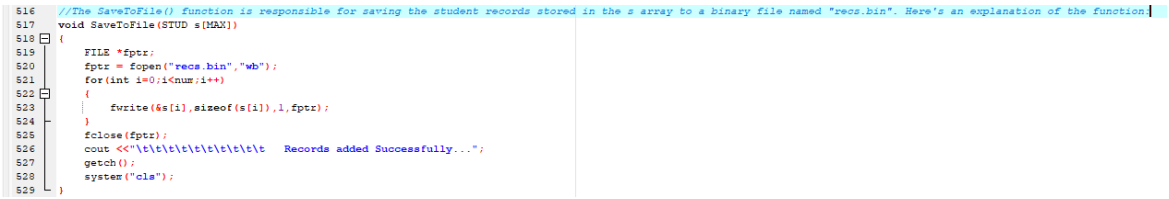
6. A message is displayed to indicate that the record was deleted successfully.

7 .If **flag** is non-zero, indicating that a record was deleted, the updated s array is saved to a file using the **SaveToFile()** function.

8. If **flag** is zero, indicating that no matching record was found, a message is displayed indicating that the record with the specified student number was not found.

9. The function then waits for user **input (getch()),** clears the screen **(system("cls")),** and pauses the execution **(system("pause")).**

---



```
516  //The SaveToFile() function is responsible for saving the student records stored in the s array to a binary file named "recs.bin". Here's an explanation of the function:
517  void SaveToFile(STUD s[MAX])
518  {
519      FILE *fptr;
520      fptr = fopen("recs.bin","wb");
521      for(int i=0;i<num;i++)
522      {
523          fwrite(&s[i],sizeof(s[i]),1,fptr);
524      }
525      fclose(fptr);
526      cout <<"\t\t\t\t\t\t\t\t\t\t\t   Records added Successfully...";
527      getch();
528      system("cls");
529  }
```

1. The function creates a file pointer **fptr** and opens the file "recs.bin" in binary write mode using the **fopen()** function. If the file does not exist, it will be created. If it already exists, its previous contents will be overwritten

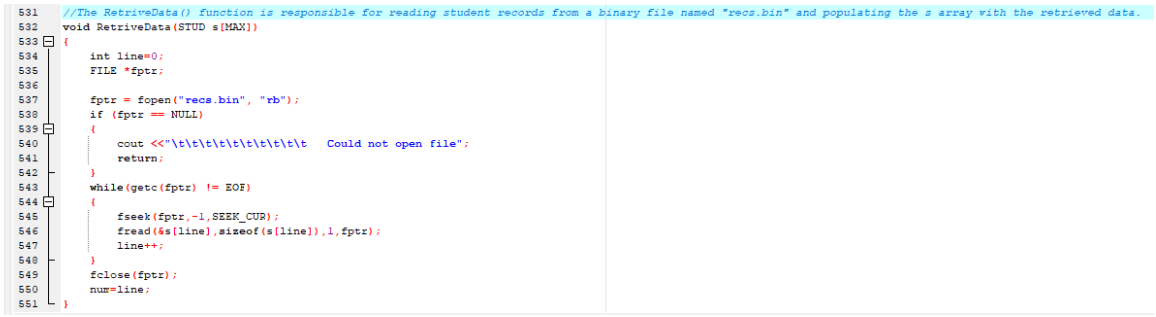
2. A loop is used to iterate through the **s** array and write each s[i] structure to the file using the **fwrite()** function. The **sizeof(s[i])** argument specifies the size of each structure to be written, and **1** indicates that only one element is being written at a time.

3. Once all the records have been written, the file is closed using the **fclose()** function.

4. A message is displayed to indicate that the records were added successfully.

5. The **getch()** function is used to wait for user input, and **system("cls")** is used to clear the screen.

---



```
531  //The RetriveData() function is responsible for reading student records from a binary file named "recs.bin" and populating the s array with the retrieved data.
532  void RetriveData(STUD s[MAX])
533  {
534      int line=0;
535      FILE *fptr;
536
537      fptr = fopen("recs.bin", "rb");
538      if (fptr == NULL)
539      {
540          cout <<"\t\t\t\t\t\t\t\t\t\t\t   Could not open file";
541          return;
542      }
543      while(getc(fptr) != EOF)
544      {
545          fseek(fptr,-1,SEEK_CUR);
546          fread(&s[line],sizeof(s[line]),1,fptr);
547          line++;
548      }
549      fclose(fptr);
550      num=line;
551  }
```

```
531  //The RetriveData() function is responsible for reading student records from a binary file named "recs.bin" and populating the s array with the retrieved data.
532  void RetriveData(STUD s[MAX])
533  {
534      int line=0;
535      FILE *fptr;
536
537      fptr = fopen("recs.bin", "rb");
538      if (fptr == NULL)
539      {
540          cout <<"\t\t\t\t\t\t\t\t\t\t\t   Could not open file";
541          return;
542      }
543      while(getc(fptr) != EOF)
544      {
545          fseek(fptr,-1,SEEK_CUR);
546          fread(&s[line],sizeof(s[line]),1,fptr);
547          line++;
548      }
549      fclose(fptr);
550      num=line;
551  }
```

1. The function declares an integer variable **line** to keep track of the current line or record being read from the file.

2. A file pointer **fptr** is created and the file **"recs.bin"** is opened in binary read mode using the **fopen()** function. If the file cannot be opened (e.g., if it doesn't exist), an error message is displayed, and the function returns.

3. The **while** loop is used to read each character from the file using getc(fptr). The loop continues until the end of the file **(EOF)** is reached.

4. Before reading the record, the file position indicator is moved back by one character using fseek(**fptr, -1, SEEK_CUR).** This is necessary because **getc()** advances the file position indicator, so we need to move it back to read the same character again.

5. The record is then read from the file and stored in the **s[line]** structure using fread**(&s[line], sizeof(s[line]), 1, fptr).** The sizeof**(s[line])** argument specifies the size of each structure to be read, and 1 indicates that only one element is read at a time.

6. After reading each record, the **line** variable is incremented to move to the next position in the s array.

7. Once all the records have been read, the file is closed using **fclose(fptr).**

8. Finally, the **num** variable is updated with the value of **line**, representing the total number of records read from the file.