STUDENT RECORD DATABASE USING SINGLE LINKEDLIST.

[YUKESH P – V23CE9Y2]

AIM OF THE PROJECT:

The aim of the project is to design and implement a dynamic student record management system using the C programming language and single linked lists. The focus is on efficiently performing various database operations while ensuring scalability, flexibility, and data persistence.

This project also aims to deepen the developer's understanding of fundamental programming concepts such as pointers, memory allocation, recursion, and file handling. Additionally, it demonstrates the practical application of linked lists in solving real-world data management problems.

ABOUT THE PROJECT:

The "Student Record Database Using C and Single Linked List" is a project designed to efficiently store and manage student records dynamically. By utilizing a single linked list, the system enables users to perform various operations on the student database, such as adding, deleting, modifying, sorting, and displaying records. This approach demonstrates the flexibility and dynamic capabilities of linked lists compared to traditional static arrays.

The integration of file handling ensures that the data is saved persistently, making it suitable for practical applications in educational institutions or organizations managing student records. The project also highlights the importance of algorithmic and data structures selection in problem solving.

OBJECTIVE:

1. Data Management: To create a system capable of efficiently storing, updating, and managing student records dynamically.

2. Practical Application of Linked Lists: To demonstrate operations such as adding, deleting, sorting, modifying, and traversing nodes in a linked list.

3. Dynamic Memory Allocation: To leverage pointers and memory allocation functions like malloc and free for efficient memory usage

4. File Handling Integration: To save and retrieve records using file handling functions (fopen, fwrite, fread, etc.), ensuring persistent data storage.

5. Error Handling: To create a robust system capable of handling invalid inputs, empty lists, and other edge cases effectively.

6. User Experience: To provide a simple and intuitive menu-driven interface for end-users.

7. Learning and Education: To serve as a project that reinforces the understanding of data structures and programming concepts.

PROJECT GOALS:

The goal of this project is to develop a dynamic and efficient student record management system using the C programming language and single linked lists, providing functionalities such as adding, deleting, modifying, and sorting records. It aims to demonstrate the practical application of linked lists in handling dynamic data, ensure data persistence through file handling, and deliver a user-friendly interface that enhances learning and highlights the importance of memory-efficient data structures in real-world scenarios.

WORKING OF THE PROJECT:

The Student Record Database Using Single Linked List operates through a menu-driven interface, enabling users to perform various tasks such as adding, deleting, modifying, sorting, and displaying student records. The key aspects of its working are outlined below:

1. Menu-Driven Interface:

The program begins with a menu that provides options for managing the database. Users can repeatedly select operations until they choose to exit.

2. Adding Records:

Users input student details such as roll number, name, and marks. A new node is dynamically created using malloc and added to the end of the linked list. If the list is empty, the new node becomes the head of the list.

3. Deleting Records:

Delete One Record: Deletes a specific record by locating it based on a unique identifier like roll number.

Delete All Records: Traverses the list, freeing all nodes, and resets the head pointer to NULL

4. Modifying Records:

Users specify the roll number of the record to modify. The program locates the node and updates its details as provided by the user.

5. Sorting Records:

Records are sorted based on user-selected criteria, such as roll number or marks.

Sorting algorithms rearrange the linked list nodes dynamically.

6. Displaying Records:

Normal Order: The list is traversed sequentially, and records are printed from the head node to the last node.

Reverse Order: The list is printed in reverse using recursion for traversal.

7. Saving and Loading Records:

Saving: Records are written to a file after each operation, ensuring data persistence.

Loading: At the start, saved records are read from the file and loaded into the linked list.

8. Error Handling and Edge Cases:

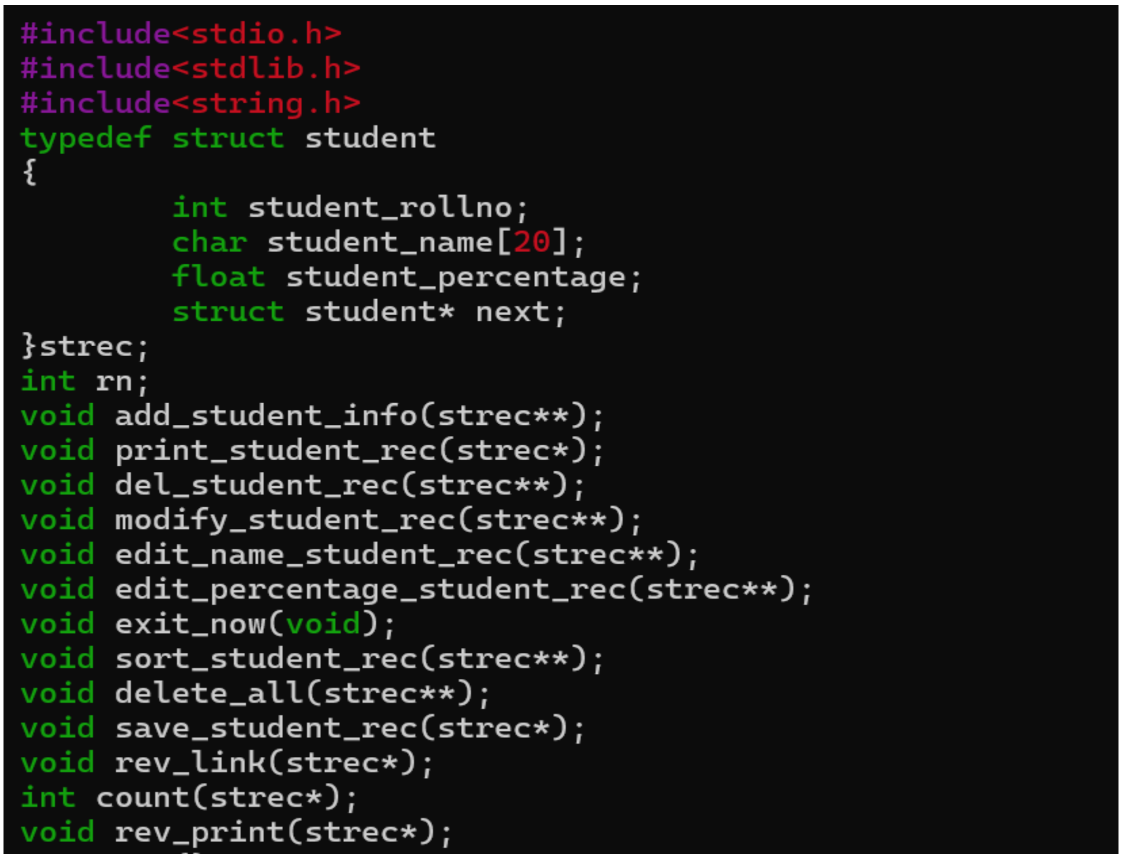
The program handles scenarios like empty lists, invalid inputs, or file errors gracefully, providing user feedback.

9. Exiting the Program:

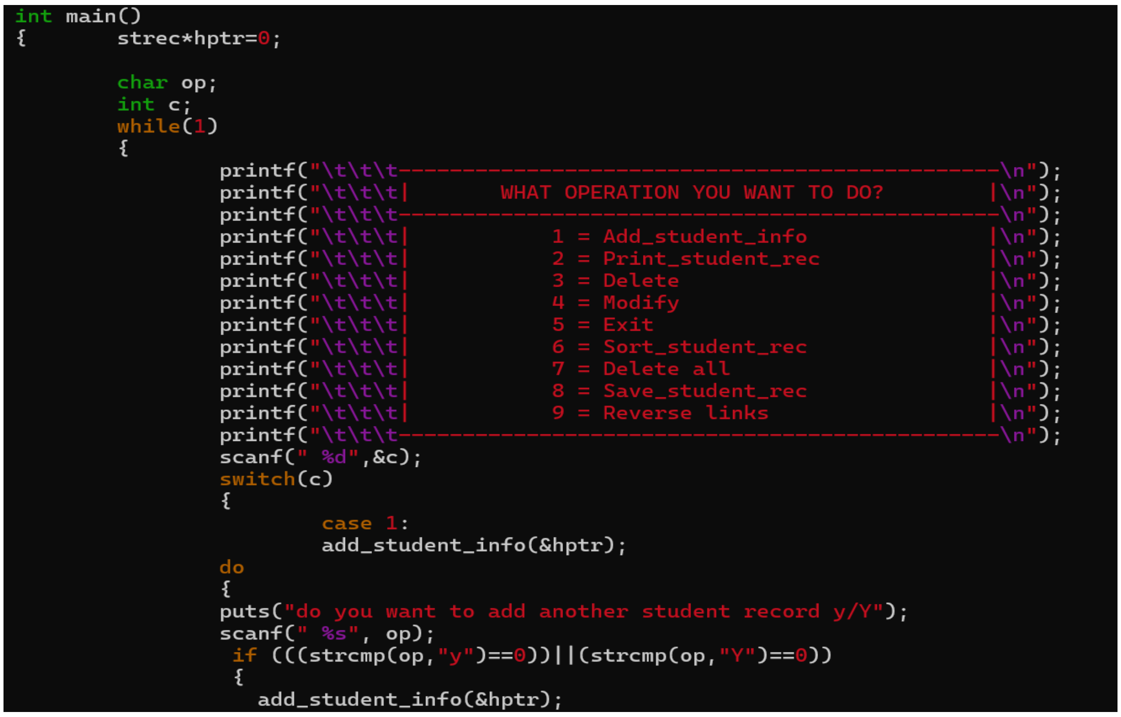
Before exiting, users are prompted to save the database, and all dynamically allocated memory is freed to prevent memory leaks.

This workflow ensures efficient data management, dynamic memory utilization, and persistent storage, providing a complete system for managing student records.

MAIN PROGRAM OF THE PROJECT:

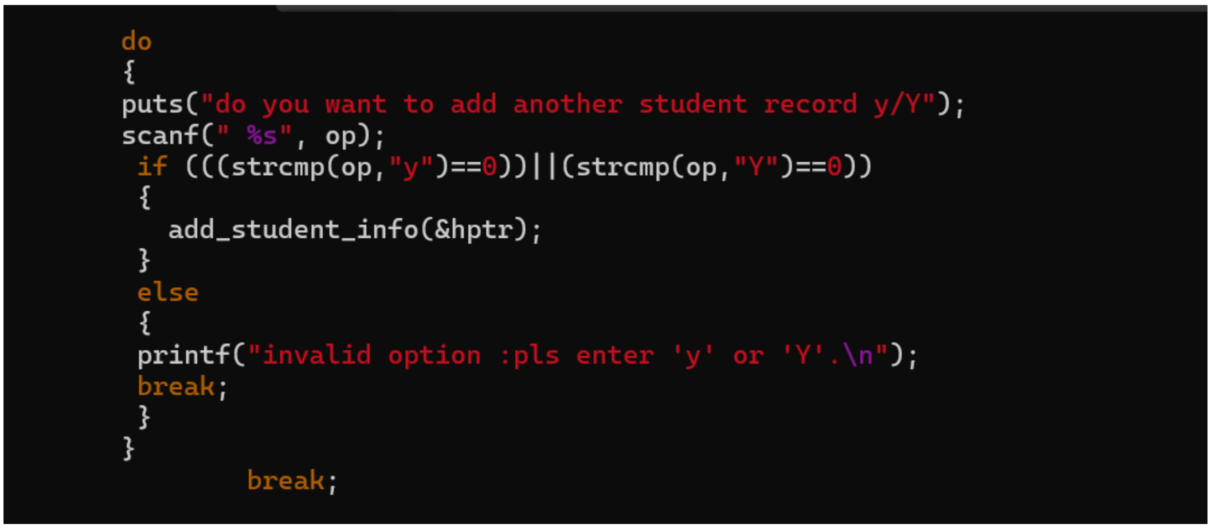


```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct student
{
        int student_rollno;
        char student_name[20];
        float student_percentage;
        struct student* next;
}strec;
int rn;
void add_student_info(strec**);
void print_student_rec(strec*);
void del_student_rec(strec**);
void modify_student_rec(strec**);
void edit_name_student_rec(strec**);
void edit_percentage_student_rec(strec**);
void exit_now(void);
void sort_student_rec(strec**);
void delete_all(strec**);
void save_student_rec(strec*);
void rev_link(strec*);
int count(strec*);
void rev_print(strec*);
```

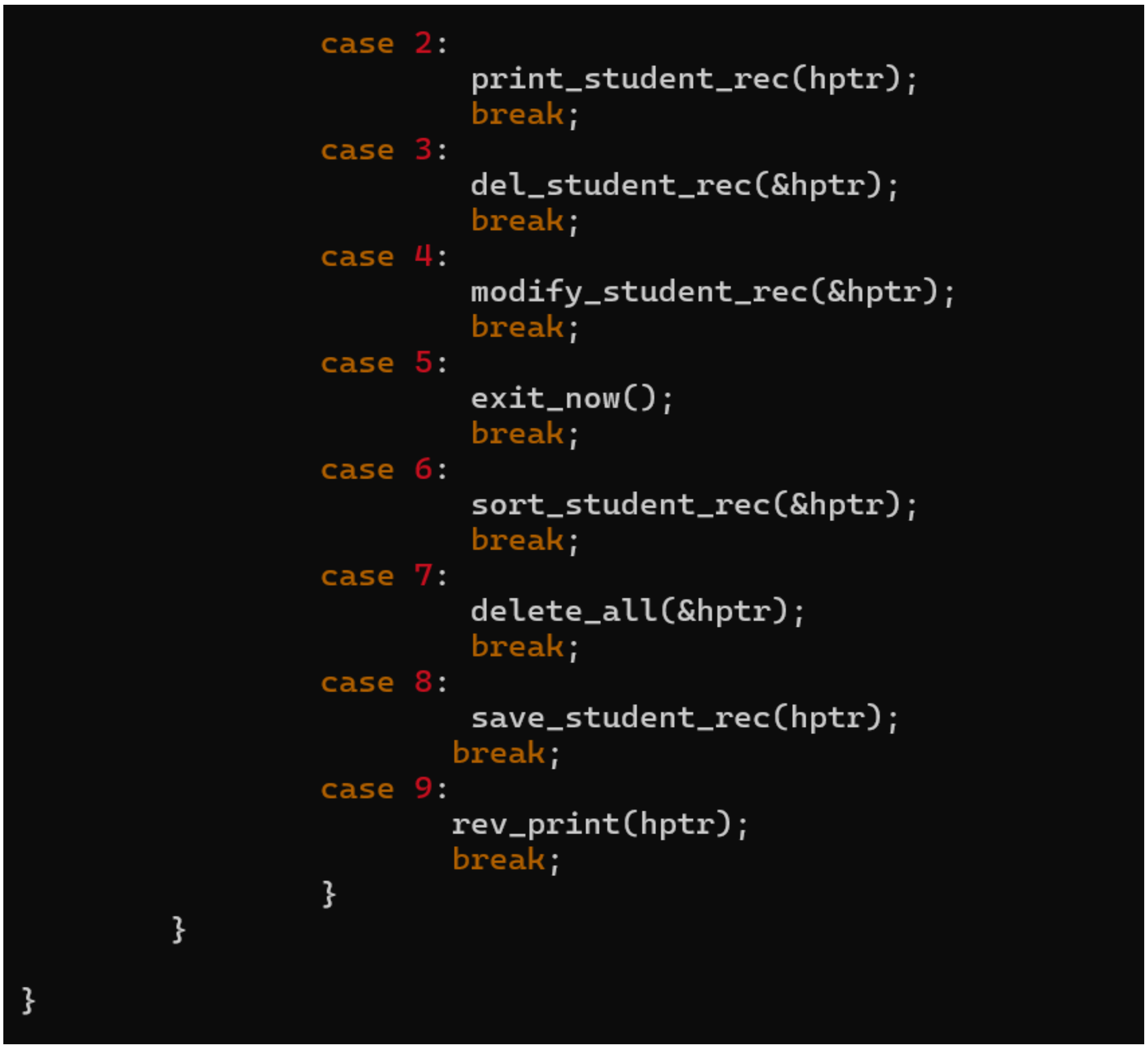


```c
int main()
{       strec*hptr=0;

        char op;
        int c;
        while(1)
        {
                printf("\t\t\t-----------------------------------------------\n");
                printf("\t\t\t|       WHAT OPERATION YOU WANT TO DO?        |\n");
                printf("\t\t\t-----------------------------------------------\n");
                printf("\t\t\t|           1 = Add_student_info              |\n");
                printf("\t\t\t|           2 = Print_student_rec             |\n");
                printf("\t\t\t|           3 = Delete                        |\n");
                printf("\t\t\t|           4 = Modify                        |\n");
                printf("\t\t\t|           5 = Exit                          |\n");
                printf("\t\t\t|           6 = Sort_student_rec              |\n");
                printf("\t\t\t|           7 = Delete all                    |\n");
                printf("\t\t\t|           8 = Save_student_rec              |\n");
                printf("\t\t\t|           9 = Reverse links                 |\n");
                printf("\t\t\t-----------------------------------------------\n");
                scanf(" %d",&c);
                switch(c)
                {
                        case 1:
                        add_student_info(&hptr);
                do
                {
                puts("do you want to add another student record y/Y");
                scanf(" %s", op);
                 if (((strcmp(op,"y")==0))||(strcmp(op,"Y")==0))
                 {
                   add_student_info(&hptr);
```



```
do
{
puts("do you want to add another student record y/Y");
scanf(" %s", op);
 if (((strcmp(op,"y")==0))||(strcmp(op,"Y")==0))
 {
   add_student_info(&hptr);
 }
 else
 {
 printf("invalid option :pls enter 'y' or 'Y'.\n");
 break;
 }
}
        break;
```



```
                case 2:
                        print_student_rec(hptr);
                        break;
                case 3:
                        del_student_rec(&hptr);
                        break;
                case 4:
                        modify_student_rec(&hptr);
                        break;
                case 5:
                        exit_now();
                        break;
                case 6:
                        sort_student_rec(&hptr);
                        break;
                case 7:
                        delete_all(&hptr);
                        break;
                case 8:
                        save_student_rec(hptr);
                       break;
                case 9:
                       rev_print(hptr);
                       break;
                }
        }

}
```

PROBLEMS FACED:

During the development of the Student Record Database Using C and Single Linked List, several challenges were encountered:

1. Memory Management:

Proper allocation and deallocation of memory using malloc and free required careful attention to prevent memory leaks or segmentation faults.

2. Handling Edge Cases:

Scenarios such as an empty list, attempting to delete non-existent records, or invalid inputs posed logical challenges that needed robust error handling.

3. File Handling Issues:

Ensuring data integrity during file read/write operations and handling errors such as missing files or incorrect file permissions were significant obstacles.

4. Debugging Recursive Functions:

Implementing reverse printing using recursion required careful debugging to avoid stack overflow and ensure correct results.

5. Sorting Linked List Nodes:

Rearranging the nodes dynamically during sorting operations, while preserving the structure of the linked list, was complex and required additional logic.

6. User Input Validation:

Preventing invalid or incomplete data entries required rigorous input validation mechanisms.

CONCLUSION:

The Student Record Database Using C and Single Linked List successfully demonstrates the application of linked lists for dynamic data management and file handling for persistent storage. The project achieves its goal of providing an efficient and user-friendly system to manage student records while showcasing the importance of memory-efficient data structures in real-world applications. Through the implementation of this project, significant insights were gained into key programming concepts such as pointers, dynamic memory allocation, recursion, and file handling. Despite challenges, the project highlights the practical potential of linked lists and serves as an educational tool for understanding data structures and algorithms.

RESULT:

The project delivers a fully functional student record management system with the following outcomes:

1. Dynamic Data Handling:

The system efficiently manages a varying number of student records without memory constraints, thanks to the use of single linked lists.

2. Comprehensive Features:

Users can add, delete, modify, sort, and display records dynamically. Sorting is achieved based on user-defined criteria like roll number or marks. Records are displayed in both normal and reverse order, demonstrating traversal techniques.

3. Data Persistence:

Data is stored persistently using file handling, ensuring that the student database is saved and retrieved across multiple sessions.

4. Robust Design:

The system handles edge cases gracefully, ensuring stability and preventing crashes due to invalid inputs or operations on empty lists.

5. User-Friendly Interface:

A menu-driven approach makes the program intuitive and easy to use, even for those with minimal technical knowledge.

6. Practical Learning:

The project provides a hands-on experience in applying theoretical knowledge of linked lists and file handling to solve real-world problems, enhancing both programming and problem-solving skills. This result demonstrates the project's success in achieving its objectives while serving as a robust example of the application of data structures and programming techniques in C.