NAME: G. GANESH

REG NO: 192373008

EXERICSE-34

Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a C program to simulate the file allocation strategy.

AIM:

To design a C program that simulates the **sequential file allocation strategy** where records are stored one after another both physically and logically, and a record can only be accessed by reading all the previous records.

Algorithm:

1. **Initialization:**
   - Define a structure to represent records in a file.
   - Create an array to simulate sequential storage.
2. **File Writing:**
   - Append new records to the end of the file.
   - Store the data in a sequential manner.
3. **File Reading:**
   - Read records sequentially from the beginning until the desired record is accessed.
   - Ensure that accessing a record requires reading all previous records.
4. **Display:**
   - Display the contents of the file sequentially.

Procedure:

1. Define a structure to represent a file record.
2. Create functions to add records, read records, and display the file content.
3. Use a loop to simulate sequential access during record retrieval.
4. Implement error handling for invalid access.

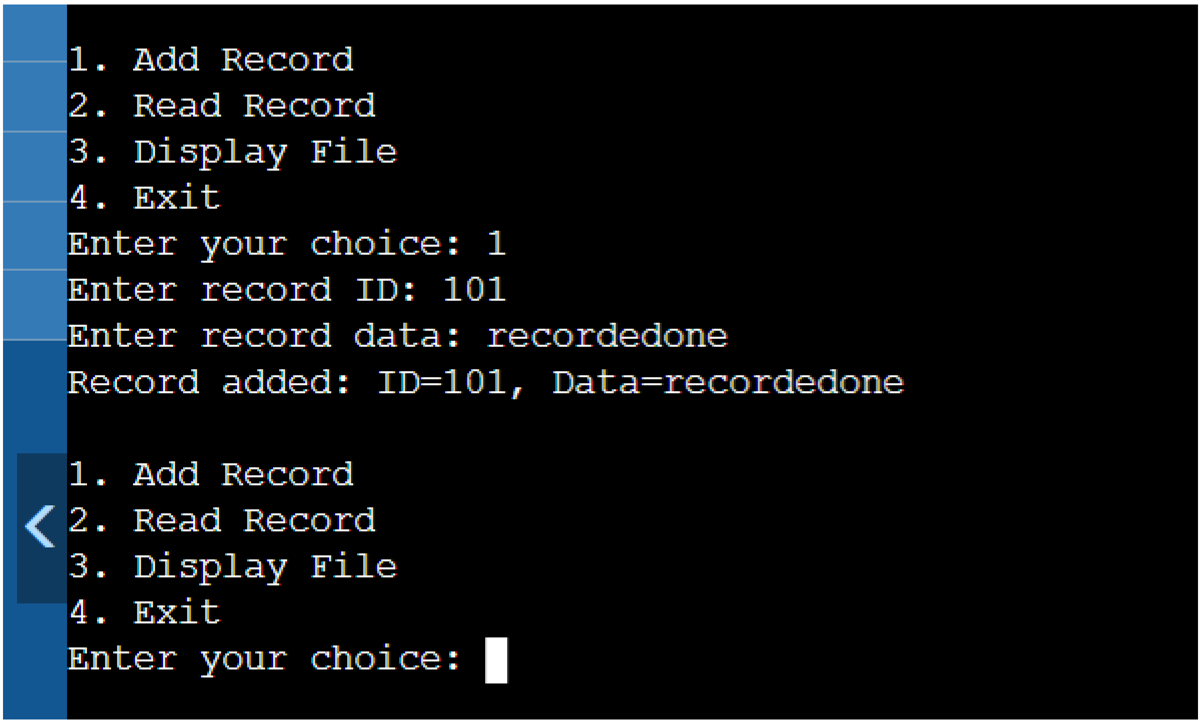
Code:

```
#include <stdio.h>

#include <string.h>

#define MAX_RECORDS 100

typedef struct {
    int id;
    char data[100];
} Record;
Record file[MAX_RECORDS];
int record_count = 0;
void add_record(int id, const char* data) {
    if (record_count >= MAX_RECORDS) {
        printf("Error: File is full.\n");
        return;
    }
    file[record_count].id = id;
    strcpy(file[record_count].data, data);
    record_count++;
    printf("Record added: ID=%d, Data=%s\n", id, data);
}
void read_record(int id) {
    printf("Reading records sequentially:\n");
    for (int i = 0; i < record_count; i++) {
        printf("Record ID=%d, Data=%s\n", file[i].id, file[i].data);
        if (file[i].id == id) {
            printf("Record found: ID=%d, Data=%s\n", file[i].id, file[i].data);
            return;
        }
    }
    printf("Record with ID=%d not found.\n", id);
}
void display_file() {
    printf("File Contents:\n");
    for (int i = 0; i < record_count; i++) {
        printf("Record ID=%d, Data=%s\n", file[i].id, file[i].data);
    }
}
int main() {
    int choice, id;
    char data[100];
    while (1) {
        printf("\n1. Add Record\n2. Read Record\n3. Display File\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter record ID: ");
                scanf("%d", &id);
                printf("Enter record data: ");
                scanf(" %[^\n]", data);
                add_record(id, data);
                break;
            case 2:
                printf("Enter record ID to read: ");
                scanf("%d", &id);
                read_record(id);
                break;
            case 3:
                display_file();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice.\n");
        }
    }
    return 0;
}
```

Result:

The program successfully simulates the **sequential file allocation strategy**, allowing records to be stored, accessed sequentially, and displayed.

Output:



```
1. Add Record
2. Read Record
3. Display File
4. Exit
Enter your choice: 1
Enter record ID: 101
Enter record data: recordedone
Record added: ID=101, Data=recordedone

1. Add Record
2. Read Record
3. Display File
4. Exit
Enter your choice:
```