REVI THIMMA REDDY-192325025

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 a **sequential file allocation strategy**, where the records of the file are stored one after another both physically and logically, and each record can only be accessed by reading all the previous records.

ALGORITHM

1. **Start**
2. Define a structure FileRecord to represent a record in the file.
3. Create an array to hold the file records and initialize them.
4. Define functions for file operations such as adding a new record, displaying all records, and accessing a specific record (sequentially).
5. Add a new record to the file at the end (sequential allocation).
6. To simulate the access strategy, iterate through all previous records before accessing the desired record.
7. Display the records as they are stored sequentially.
8. **Stop**

PROCEDURE

1. Include necessary libraries (stdio.h for input/output and stdlib.h for memory allocation).
2. Define a FileRecord structure to represent a file record.
3. Create a function addRecord() to simulate the addition of new records to the file.
4. Create a function displayRecords() to display the current file records sequentially.

5. Create a function accessRecord() to simulate sequential access by reading previous records.
6. Initialize the file and perform operations like adding records and displaying or accessing them sequentially.
7. **End**

CODE:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_RECORDS 100

typedef struct {
  int id;
  char data[100];
} FileRecord;

FileRecord file[MAX_RECORDS];
int recordCount = 0;

void addRecord(int id, const char *data) {
  if (recordCount < MAX_RECORDS) {
    file[recordCount].id = id;
    snprintf(file[recordCount].data, sizeof(file[recordCount].data), "%s", data);
    recordCount++;
  } else {
    printf("File is full, cannot add more records.\n");
  }
}
```

```
void displayRecords() {
  if (recordCount == 0) {
    printf("No records in the file.\n");
    return;
  }

  for (int i = 0; i < recordCount; i++) {
    printf("Record %d: %s\n", file[i].id, file[i].data);
  }
}

void accessRecord(int recordId) {
  if (recordId > recordCount || recordId < 1) {
    printf("Record not found.\n");
    return;
  }

  printf("Accessing Record %d: %s\n", file[recordId - 1].id, file[recordId - 1].data);
}

int main() {
  int choice, id;
  char data[100];

  while (1) {
    printf("\nFile Allocation System\n");
    printf("1. Add Record\n");
    printf("2. Display All Records\n");
```

```
printf("3. Access a Specific Record\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
getchar();

switch (choice) {
  case 1:
    printf("Enter Record ID: ");
    scanf("%d", &id);
    getchar();
    printf("Enter Record Data: ");
    fgets(data, sizeof(data), stdin);
    data[strcspn(data, "\n")] = 0;
    addRecord(id, data);
    break;
  case 2:
    displayRecords();
    break;
  case 3:
    printf("Enter Record ID to access: ");
    scanf("%d", &id);
    accessRecord(id);
    break;
  case 4:
    exit(0);
  default:
    printf("Invalid choice. Please try again.\n");
```

```
    }
  }

  return 0;
}
```

OUTPUT: