The C-Based Movie Search Engine

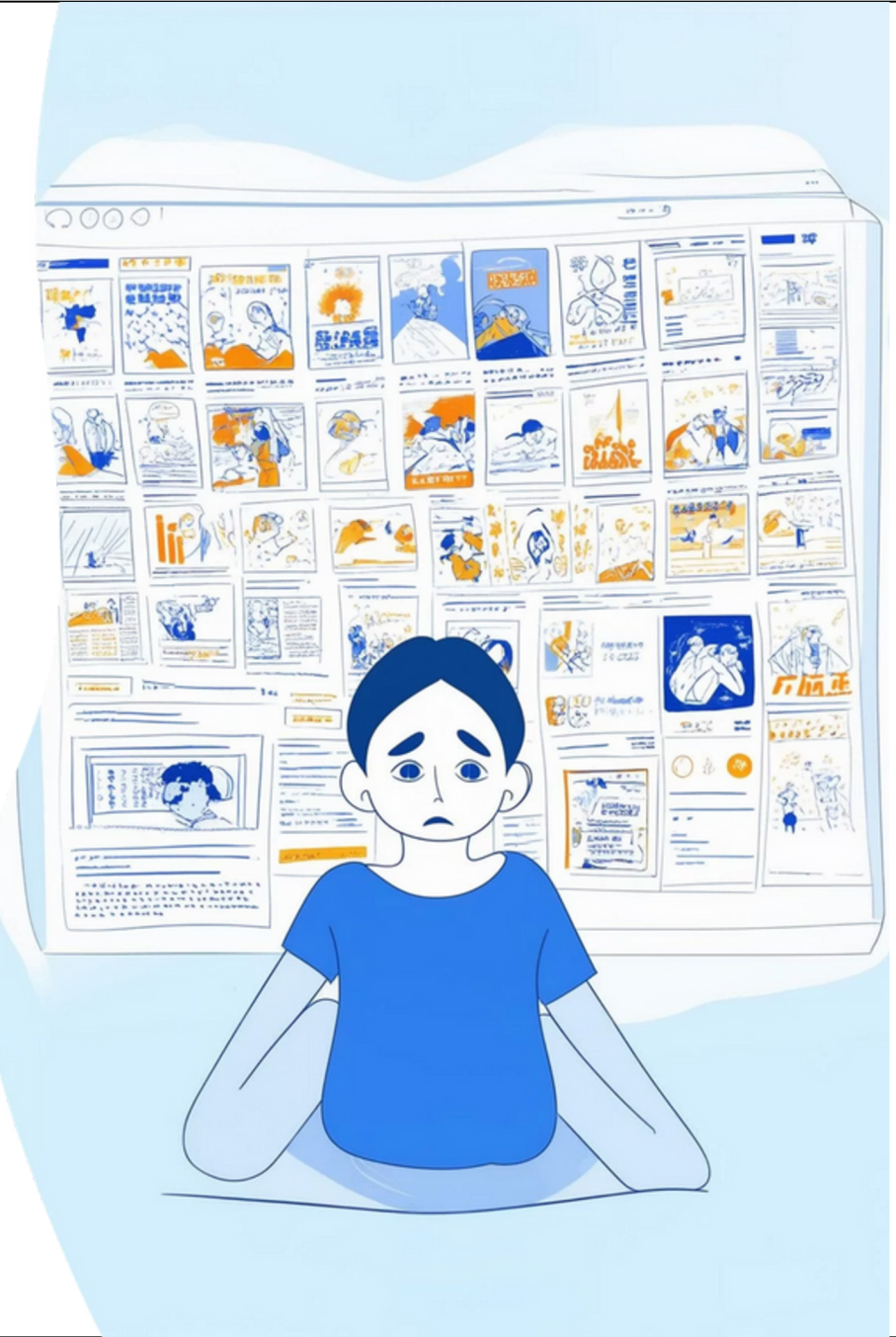
By:Khushi Tyagi,AsimShaik, Vedant, Prashant,Tanuj

Welcome to our presentation on a new movie search engine, built from the ground up in C. Designed for efficiency and ease of use, this project aims to simplify movie discovery for every enthusiast.

Meet the Members&

Eachmemberplayedacrucialroleinbringingthismoviesearch engine to life.

Name	Enrollment	Responsibilities
Khushi Tyagi	2403031461259	Project Lead, Data Structure Implementation
Asim Shaik	2403031460813	Algorithm Design & Optimization
Prashant Kumar Singh	2403031460602	User Interface & Input Handling
Tanuj Kumar Singh	2403031461745	Testing & Debugging
Vedant Mali	2403031461398	Documentation & Presentation Support



The Search for Simplicity

> "Everymovie lover knows the struggle 4dozensofplatforms, endless scrolling, and no simple way to find the right film at the right time. Existing search options are scattered, inconsistent, and time-consuming. Users need one intelligent engine that can instantly search, filter, and rank movies in a single place."

This problem statement highlights a universal frustration: the difficulty of efficiently finding movies across fragmented digital landscapes. We set out to create a unified, robust solution.

Our Solution

Introducing the C-Based Movie Search Engine

We developed a robust movie search engine implemented entirely in C, designed for efficiency and ease of use. This application provides core functionalities crucial for any movie enthusiast.

Add & Display Movies

Effortlessly inputnewmoviedetails and view the full library.

Search by Name

Quicklylocatefilmsusing their titles with precise results.

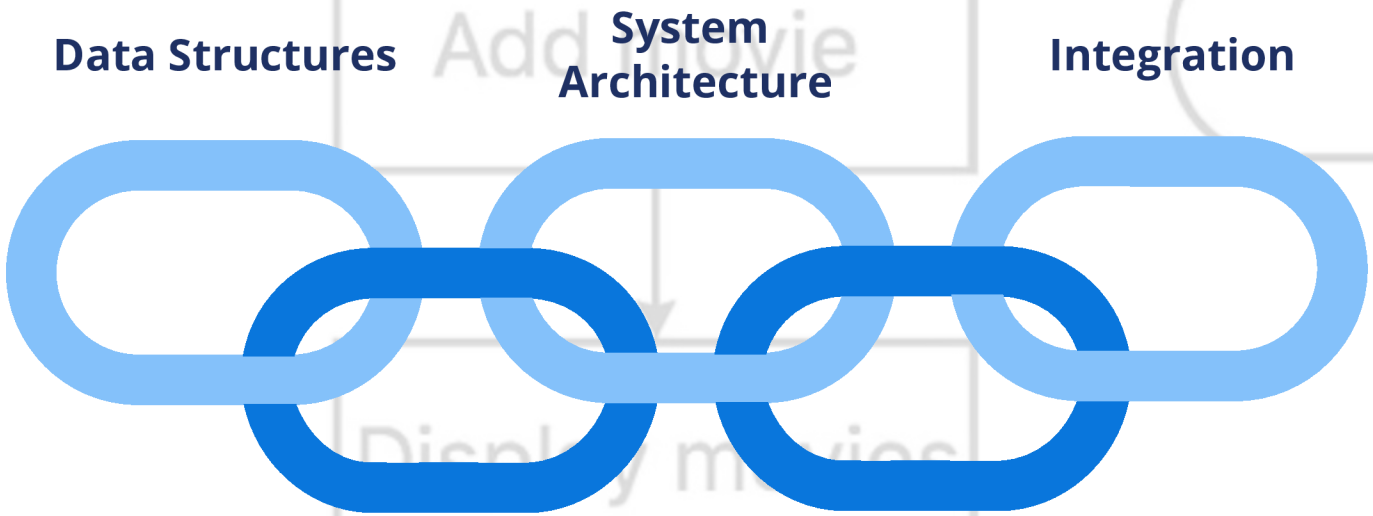
Sort by Year & Rating

Organize moviesby releaseyear or audience rating for easy browsing.

The Blueprint of Our Engine

Thearchitecture ofourC-basedMovie SearchEngineisbuiltonafoundation of carefully selected data structures and algorithms, ensuring optimal performanceanduser experience.

This card gives an overview, with subsequent cards delving into the specifics of how we organize and process movie data.



Data Structures Powering the Engine

Thefoundationofourmoviesearch enginereliesonfundamental Cdatastructures. We utilize an **Array of Structures** to store movie information, offering a structured and efficient way to manage diverse data types for each film.

This approach allows us to group related movie attributes like ID, name, year, and rating into a single, cohesive unit.

```
typedef struct {
    int id;
    char name[100];
    int year;
    float rating;
} Movie;
```

Each `Movie` struct serves as a blueprint for a single movie entry, making data access and manipulation intuitive. The array then holds multiple instances of this structure, forming our movie database.

Algorithms: How We Find Your Favorites

Ourengineemploysclassicalgorithmstoperform search andsort operationsefficiently. For searching movies by name, we implement a **Linear Search** algorithm.

Linear search iterates through each element until a match is found, a simple yet effective method for smaller datasets or unsorted lists.

```
for (int i = 0; i < count; i++) {
  if (strcmp(movies[i].name, target) == 0) {
    printf("Movie found: %s\n", movies[i].name);
  }
}
```

This snippet demonstrates how the search function compares the target movie name with each entry in our movie array. While effective, we'll explore optimizations for larger databases later.

The application also features a simple menu-driven interface, built around a do-while loop, allowing users to interact with the movie database functions intuitively.

```
do {
 printf("1. Add\n2. Display\n3. Search\n4. Sort\n0. Exit\n");
 scanf("%d", &choice);
 switch(choice) { ... }
} while(choice != 0);
```

Complete Movie Search Engine Code

```
#include <stdio.h>
#include <string.h> // Include for strcmp and strcspn
#define MAX 100

typedef struct {
 int id;
 char name[100];
 int year;
 float rating;
} Movie;

Movie movies[MAX];
int count = 0;

// Function to add a movie
void addMovie() {
 if (count >= MAX) {
 printf("Database is full!\n");
 return;
 }
 printf("Enter Movie ID: ");
 scanf("%d", &movies[count].id);
 printf("Enter Movie Name: ");
 getchar(); // To consume the newline
 fgets(movies[count].name, 100, stdin);
 movies[count].name[strcspn(movies[count].name, "\n")] = '\0'; // Remove newline
 printf("Enter Release Year: ");
 scanf("%d", &movies[count].year);
 printf("Enter Rating (0.0 - 10.0): ");
 scanf("%f", &movies[count].rating);
 count++;
 printf("Movie added successfully.\n");
}

// Function to display all movies
void displayMovies() {
 if (count == 0) {
 printf("No movies to display.\n");
 return;
 }
 printf("\n%-5s %-30s %-10s %-10s\n", "ID", "Name", "Year", "Rating");
 printf("-----------------------------------------------------------\n");
 for (int i = 0; i < count; i++) {
 printf("%-5d %-30s %-10d %-10.1f\n", movies[i].id, movies[i].name, movies[i].year, movies[i].rating);
 }
}

// Linear search by name
void searchByName() {
 char target[100];
 printf("Enter movie name to search: ");
 getchar();
 fgets(target, 100, stdin);
 target[strcspn(target, "\n")] = '\0';
 int found = 0;
 for (int i = 0; i < count; i++) {
 if (strcmp(movies[i].name, target) == 0) {
 printf("Movie found: ID: %d, Name: %s, Year: %d, Rating: %.1f\n", movies[i].id, movies[i].name, movies[i].year, movies[i].rating);
 found = 1;
 }
 }
 if (!found) {
 printf("Movie not found.\n");
 }
}

// Sort by year (ascending) using Insertion Sort
void sortByYear() {
 for (int i = 1; i < count; i++) {
 Movie key = movies[i];
 int j = i - 1;
 while (j >= 0 && movies[j].year > key.year) {
 movies[j + 1] = movies[j];
 j--;
 }
 movies[j + 1] = key;
 }
}

// Binary search by year (after sorting)
void searchByYear() {
 if (count == 0) {
 printf("No movies to search.\n");
 return;
 }
 sortByYear(); // Ensure it's sorted
 int targetYear;
 printf("Enter release year to search: ");
 scanf("%d", &targetYear);
 int low = 0, high = count - 1, found = 0;
 while (low <= high) {
 int mid = (low + high) / 2;
 if (movies[mid].year == targetYear) {
 // Display all matching years
 int i = mid;
 while (i >= 0 && movies[i].year == targetYear) i--;
 i++;
 printf("Movies released in %d:\n", targetYear);
 while (i < count && movies[i].year == targetYear) {
 printf("ID: %d, Name: %s, Rating: %.1f\n", movies[i].id, movies[i].name, movies[i].rating);
 i++;
 }
 found = 1;
 break;
 } else if (movies[mid].year < targetYear) {
 low = mid + 1;
 } else {
 high = mid - 1;
 }
 }
 if (!found) {
 printf("No movies found for year %d.\n", targetYear);
 }

}

// Sort by rating (descending) using Bubble Sort
void sortByRating() {
 for (int i = 0; i < count - 1; i++) {
 for (int j = 0; j < count - i - 1; j++) {
 if (movies[j].rating < movies[j + 1].rating) {
 Movie temp = movies[j];
 movies[j] = movies[j + 1];
 movies[j + 1] = temp;
 }
 }
 }
 printf("Movies sorted by rating (descending).\n");
}

int main() {
int choice;
do {
printf("\n===== Movie Database Menu =====\n");
printf("1. Add Movie\n");
printf("2. Display Movies\n");
printf("3. Search by Name\n");
printf("4. Search by Year\n");
printf("5. Sort by Rating\n");
printf("6. Sort by Year\n");
printf("0. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1: addMovie(); break;
case 2: displayMovies(); break;
case 3: searchByName(); break;
case 4: searchByYear(); break;
case 5: sortByRating(); displayMovies(); break;
case 6: sortByYear(); displayMovies(); break;
case 0: printf("Exiting Movie Database. Goodbye!\n"); break;
default: printf("Invalid choice. Try again.\n");
}
} while (choice != 0);
return 0;

}
```

Application Demonstration & Test Cases

To illustrate the functionality of our C-based movie search engine, let's walk through some typical user interactions and their corresponding outputs, followed by a summary of key test cases.

Input Example 1: Add & Display Movies

Add "Inception" (2010, 8.8) and "Interstellar" (2014, 8.6).

```
Enter Movie ID: 101
Enter Movie Name: Inception
Enter Release Year: 2010
Enter Rating (0.0 - 10.0): 8.8
Enter Movie ID: 102
Enter Movie Name: Interstellar
Enter Release Year: 2014
Enter Rating (0.0 - 10.0): 8.6

ID Name                    Year      Rating
-------------------------------------------------------
101 Inception              2010      8.8
102 Interstellar           2014      8.6
```

Input Example 2: Search by Name

Search for "Inception".

```
Enter movie name to search: Inception

Movie found: ID: 101, Name: Inception, Year: 2010, Rating: 8.8
```

Input Example 3: Search by Year

Search for movies released in 2014.

```
Enter release year to search: 2014

Movies released in 2014:
ID: 102, Name: Interstellar, Rating: 8.6
```

Input Example 4: Sort by Rating

Sort existing movies by rating in descending order.

```
Movies sorted by rating (descending).
```

This section provides a practical demonstration of our C-based movie search engine's capabilities through typical user interactions and their corresponding outputs. We'll walk through adding, displaying, searching, and sorting movies.

Input Example 4: Sort by Rating

Output after sorting movies by rating in descending order:

```
Movies sorted by rating (descending).

ID Name                    Year      Rating
-----------------------------------------------------
101 Inception              2010      8.8
102 Interstellar           2014      8.6
```

Input Example 5: Sort by Year

Sort existing movies by release year in ascending order.

```
ID Name                    Year      Rating
-----------------------------------------------------
101 Inception              2010      8.8
102 Interstellar           2014      8.6
```

Test Cases

Case	Input	Operation	Expected Output
1	Add 2 movies (Inception, Interstellar) Search by Name = "Inception" Search by Year = 2014	Display Movies	Both movies displayed in tabular format Inception details shown
2		Search	
3	Sort by Rating	Search	Interstellar details shown
4		Sort	Movies ordered