**SOURCE** **CODE** **::**

**BY~**

**G. ABHINAY [AP23110011114]**

**T. SWETHA PRIYA [AP23110011116]**

**A. MANOJ [AP23110011118]**

**G. VISHNU VARDHAN [AP23110011138]**

**MODULE NAME :: UNIVERSITY**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MAX\_UNIVERSITIES 100

#define MAX\_STRING\_LENGTH 100

// Define the structure for university records

typedef struct {

int id;

char univ\_code[MAX\_STRING\_LENGTH];

char univ\_name[MAX\_STRING\_LENGTH];

char univ\_address[MAX\_STRING\_LENGTH];

char univ\_email[MAX\_STRING\_LENGTH];

char univ\_website[MAX\_STRING\_LENGTH];

} University;

// Function prototypes

void university\_create(University universities[], int \*count);

void university\_update(University universities[], int count);

void university\_retrieve(University universities[], int count);

void university\_delete(University universities[], int \*count);

void university\_mergesort(University universities[], int left, int right);

void merge(University universities[], int left, int mid, int right);

void university\_quicksort(University universities[], int low, int high);

int partition(University universities[], int low, int high);

int university\_linearsearch(University universities[], int count, int id);

int university\_binarysearch(University universities[], int left, int right, int id);

void university\_storing(University universities[], int count);

void university\_compare\_linearsearch(University universities[], int count, int id);

void university\_compare\_binarysearch(University universities[], int count, int id);

void university\_compare\_quicksort(University universities[], int count);

void university\_compare\_mergesort(University universities[], int count);

void university\_complexity\_linearsearch(int n);

void university\_complexity\_binarysearch(int n);

void university\_complexity\_mergesort(int n);

void university\_complexity\_quicksort(int n);

void writeToFile(University universities[], int count);

int main() {

University universities[MAX\_UNIVERSITIES];

int count = 0;

int choice;

do {

printf("\nUniversity Management System\n");

printf("1. Create University\n");

printf("2. Update University\n");

printf("3. Retrieve Universities\n");

printf("4. Delete University\n");

printf("5. Sort Universities (Merge Sort)\n");

printf("6. Sort Universities (Quick Sort)\n");

printf("7. Search University (Linear Search)\n");

printf("8. Search University (Binary Search)\n");

printf("9. Store Universities to File\n");

printf("10. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

university\_create(universities, &count);

break;

case 2:

university\_update(universities, count);

break;

case 3:

university\_retrieve(universities, count);

break;

case 4:

university\_delete(universities, &count);

break;

case 5:

university\_compare\_mergesort(universities, count);

break;

case 6:

university\_compare\_quicksort(universities, count);

break;

case 7: {

int id;

printf("Enter university ID to search: ");

scanf("%d", &id);

university\_compare\_linearsearch(universities, count, id);

break;

}

case 8: {

int id;

printf("Enter university ID to search: ");

scanf("%d", &id);

university\_compare\_binarysearch(universities, count, id);

break;

}

case 9:

university\_storing(universities, count);

break;

case 10:

printf("Exiting...\n");

break;

default:

printf("Invalid choice! Please try again.\n");

}

} while (choice != 10);

return 0;

}

// Function implementations

void university\_create(University universities[], int \*count) {

if (\*count >= MAX\_UNIVERSITIES) {

printf("Cannot add more universities.\n");

return;

}

University u;

printf("Enter ID: ");

scanf("%d", &u.id);

printf("Enter University Code: ");

scanf("%d", &u.univ\_code);

printf("Enter University Name: ");

scanf(" %s", &u.univ\_name); // To read string with spaces

printf("Enter University Address: ");

scanf(" %s", &u.univ\_address);

printf("Enter University Email: ");

scanf("%s", &u.univ\_email);

printf("Enter University Website: ");

scanf("%s", &u.univ\_website);

universities[\*count] = u;

(\*count)++;

printf ("University added successfully!\n");

}

void university\_update(University universities[], int count) {

int id, found = 0;

printf("Enter ID of the university to update: ");

scanf("%d", &id);

for (int i = 0; i < count; i++) {

if (universities[i].id == id) {

found = 1;

printf("Updating details for ID %d:\n", id);

printf("Enter new University Code: ");

scanf("%d", universities[i].univ\_code);

printf("Enter new University Name: ");

scanf(" %s", &universities[i].univ\_name);

printf("Enter new University Address: ");

scanf(" %s", &universities[i].univ\_address);

printf("Enter new University Email: ");

scanf("%s", &universities[i].univ\_email);

printf("Enter new University Website: ");

scanf("%s", &universities[i].univ\_website);

printf("University updated successfully!\n");

break;

}

}

if (!found) {

printf("University with ID %d not found.\n", id);

}

}

void university\_retrieve(University universities[], int count) {

printf("List of Universities:\n");

for (int i = 0; i < count; i++) {

printf("ID: %d, Code: %d, Name: %s, Address: %s, Email: %s, Website: %s\n",

universities[i].id, universities[i].univ\_code, universities[i].univ\_name,

universities[i].univ\_address, universities[i].univ\_email, universities[i].univ\_website);

}

}

void university\_delete(University universities[], int \*count) {

int id, found = 0;

printf("Enter ID of the university to delete: ");

scanf("%d", &id);

for (int i = 0; i < \*count; i++) {

if (universities[i].id == id) {

found = 1;

for (int j = i; j < \*count - 1; j++) {

universities[j] = universities[j + 1];

}

(\*count)--;

printf("University with ID %d deleted successfully!\n", id);

break;

}

}

if (!found) {

printf("University with ID %d not found.\n", id);

}

}

void university\_mergesort(University universities[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

university\_mergesort(universities, left, mid);

university\_mergesort(universities, mid + 1, right);

merge(universities, left, mid, right);

}

}

void merge(University universities[], int left, int mid, int right) {

int i, j, k;

int n1 = mid - left + 1;

int n2 = right - mid;

University \*L = (University \*)malloc(n1 \* sizeof(University));

University \*R = (University \*)malloc(n2 \* sizeof(University));

for (i = 0; i < n1; i++)

L[i] = universities[left + i];

for (j = 0; j < n2; j++)

R[j] = universities[mid + 1 + j];

i = 0;

j = 0;

k = left;

while (i < n1 && j < n2) {

if (L[i].id <= R[j].id) {

universities[k] = L[i];

i++;

} else {

universities[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

universities[k] = L[i];

i++;

k++;

}

while (j < n2) {

universities[k] = R[j];

j++;

k++;

}

free(L);

free(R);

}

void university\_quicksort(University universities[], int low, int high) {

if (low < high) {

int pi = partition(universities, low, high);

university\_quicksort(universities, low, pi - 1);

university\_quicksort(universities, pi + 1, high);

}

}

int partition(University universities[], int low, int high) {

int pivot = universities[high].id;

int i = (low - 1);

for (int j = low; j < high; j++) {

if (universities [j].id < pivot) {

i++;

University temp = universities[i];

universities[i] = universities[j];

universities[j] = temp;

}

}

University temp = universities[i + 1];

universities[i + 1] = universities[high];

universities[high] = temp;

return (i + 1);

}

int university\_linearsearch(University universities[], int count, int id) {

for (int i = 0; i < count; i++) {

if (universities[i].id == id) {

return i; // Return the index if found

}

}

return -1; // Return -1 if not found

}

int university\_binarysearch(University universities[], int left, int right, int id) {

while (left <= right) {

int mid = left + (right - left) / 2;

if (universities[mid].id == id) {

return mid; // Return the index if found

}

if (universities[mid].id < id) {

left = mid + 1; // Search in the right half

} else {

right = mid - 1; // Search in the left half

}

}

return -1; // Return -1 if not found

}

void university\_storing(University universities[], int count) {

writeToFile(universities, count);

}

void writeToFile(University universities[], int count) {

FILE \*file = fopen("university\_setting.txt", "w");

if (file == NULL) {

printf("Error opening file!\n");

return;

}

for (int i = 0; i < count; i++) {

fprintf(file, "ID: %d, Code: %s, Name: %s, Address: %s, Email: %s, Website: %s\n",

universities[i].id, universities[i].univ\_code, universities[i].univ\_name,

universities[i].univ\_address, universities[i].univ\_email, universities[i].univ\_website);

}

fclose(file);

printf("Data stored successfully in university\_setting.txt\n");

}

void university\_compare\_linearsearch(University universities[], int count, int id) {

clock\_t start = clock();

int index = university\_linearsearch(universities, count, id);

clock\_t end = clock();

if (index != -1) {

printf("University found at index %d\n", index);

} else {

printf("University with ID %d not found.\n", id);

}

university\_complexity\_linearsearch(count);

double time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Linear Search Time: %f seconds\n", time\_taken);

}

void university\_compare\_binarysearch(University universities[], int count, int id) {

university\_quicksort(universities, 0, count - 1); // Ensure the array is sorted

clock\_t start = clock();

int index = university\_binarysearch(universities, 0, count - 1, id);

clock\_t end = clock();

if (index != -1) {

printf("University found at index %d\n", index);

} else {

printf("University with ID %d not found.\n", id);

}

university\_complexity\_binarysearch(count);

double time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Binary Search Time: %f seconds\n", time\_taken);

}

void university\_compare\_quicksort(University universities[], int count) {

clock\_t start = clock();

university\_quicksort(universities, 0, count - 1);

clock\_t end = clock();

printf("Universities sorted using Quick Sort.\n");

university\_complexity\_quicksort(count);

double time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Quick Sort Time: %f seconds\n", time\_taken);

}

void university\_compare\_mergesort(University universities[], int count) {

clock\_t start = clock();

university\_mergesort(universities, 0, count - 1);

clock\_t end = clock();

printf("Universities sorted using Merge Sort.\n");

university\_complexity\_mergesort(count);

double time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Merge Sort Time: %f seconds\n", time\_taken);

}

void university\_complexity\_linearsearch(int n) {

printf("Time Complexity of Linear Search: O(n)\n");

}

void university\_complexity\_binarysearch(int n) {

printf("Time Complexity of Binary Search: O(log n )\n");

}

void university\_complexity\_mergesort(int n) {

printf("Time Complexity of Merge Sort: O(n log n)\n");

}

void university\_complexity\_quicksort(int n) {

printf("Time Complexity of Quick Sort: O(n log n) on average, O(n^2) in the worst case\n");

}