```cpp
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
#include<time.h>
#include<ctype.h>

/* GIANT NOTE:
    Section 1:
    - Recursive (Take a value #1 from array)
    - Struct
    - Search
    - Sort

    Section 2:
    - Insert
    - Update
    - Sort every insert
    - Write save file
*/

/* Small Note
Sorting direction!:
    - < Descending, high to low
    - > Ascending, low to high
    this is when the IF comparison.
    The IF it means, was when comparing with the DATA in array themselves.
*/

struct Datas {
    char name[100];
    int age;
    bool gender; //male = true
    float fill;
} BioDatas[1000];

//Base functions
void swapy(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void swapStruct(struct Datas *InfoA, struct Datas *InfoB) {
    struct Datas temps = *InfoA;
    *InfoA = *InfoB;
    *InfoB = temps;
}
// end base functions

//Install! Sort Algorithms. collected by Geeks for geeks
//Begin Sort sets
// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)

        // Last i elements are already in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swapy(&arr[j], &arr[j + 1]);
}

void AgebubbleSort(struct Datas informations[], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (informations[j].age < informations[j + 1].age) { //Descending
              <
                swapStruct(&informations[j], &informations[j + 1]);
            }
        }
    }
}

void NamebubbleSort(struct Datas informations[], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (strcmp(informations[j].name, informations[j + 1].name) > 0)
              { //Ascending >
                swapStruct(&informations[j], &informations[j + 1]);
            }
        }
    }
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n - 1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i + 1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

        // Swap the found minimum element with the first element
        swapy(&arr[min_idx], &arr[i]);
    }
}

void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        /* Move elements of arr[0..i-1], that are
        greater than key, to one position ahead
        of their current position */
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

//Merge Sort begin
// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
//void merge(int arr[], int l, int m, int r)
//{
//  int i, j, k;
//  int n1 = m - l + 1;
//  int n2 = r - m;
//
//  /* create temp arrays */
//  int L[n1], R[n2];
//
//  /* Copy data to temp arrays L[] and R[] */
//  for (i = 0; i < n1; i++)
//      L[i] = arr[l + i];
//  for (j = 0; j < n2; j++)
//      R[j] = arr[m + 1 + j];
//
//  /* Merge the temp arrays back into arr[l..r]*/
//  i = 0; // Initial index of first subarray
//  j = 0; // Initial index of second subarray
//  k = l; // Initial index of merged subarray
//  while (i < n1 && j < n2)
//  {
//      if (L[i] <= R[j])
//      {
//          arr[k] = L[i];
//          i++;
//      }
//      else
//      {
//          arr[k] = R[j];
//          j++;
//      }
//      k++;
//  }
//
//  /* Copy the remaining elements of L[], if there
//  are any */
//  while (i < n1)
//  {
//      arr[k] = L[i];
//      i++;
//      k++;
//  }
//
//  /* Copy the remaining elements of R[], if there
//  are any */
//  while (j < n2)
//  {
//      arr[k] = R[j];
//      j++;
//      k++;
//  }
//}
//
///* l is for left index and r is right index of the
//sub-array of arr to be sorted */
//void mergeSort(int arr[], int l, int r)
//{
//  if (l < r)
//  {
//      // Same as (l+r)/2, but avoids overflow for
//      // large l and h
//      int m = l + (r - l) / 2;
//
//      // Sort first and second halves
//      mergeSort(arr, l, m);
//      mergeSort(arr, m + 1, r);
//
//      merge(arr, l, m, r);
//  }
//}
//Merge Sort Ends

//Quick Sort Begin
/* This function takes last element as pivot, places
the pivot element at its correct position in sorted
array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition(int arr[], int low, int high)
{
    int pivot = arr[high];    // pivot
    int i = (low - 1);  // Index of smaller element

    for (int j = low; j <= high - 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;    // increment index of smaller element
            swapy(&arr[i], &arr[j]);
        }
    }
    swapy(&arr[i + 1], &arr[high]);
    return (i + 1);
```

```cpp
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low  --> Starting index,
high  --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
        at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
//Quick Sort Ended

//End Sort sets

//Install! Search Algorithms. collected by Geeks for geeks
//Begin Search sets
// Linearly search x in arr[].  If x is present then return its
// location,  otherwise return -1
int linearSearch(int arr[], int n, int x)
{
    int i;
    for (i = 0; i<n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

int NamelinearSearch(struct Datas informations[], int n, char name[]) {
    for (int i = 0; i < n; i++) {
        if (strcmp(informations[i].name, name) == 0) {
            return i;
        }
    }
    return -1;
}

int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l)
    {
        int mid = l + (r - l) / 2;

        // If the element is present at the middle
        // itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    // We reach here when element is not
    // present in array
    return -1;
}

// If x is present in arr[0..n-1], then returns
// index of it, else returns -1.
int interpolationSearch(int arr[], int n, int x)
{
    // Find indexes of two corners
    int lo = 0, hi = (n - 1);

    // Since array is sorted, an element present
    // in array must be in range defined by corner
    while (lo <= hi && x >= arr[lo] && x <= arr[hi])
    {
        // Probing the position with keeping
        // uniform distribution in mind.
        int pos = lo + (((double)(hi - lo) /
            (arr[hi] - arr[lo]))*(x - arr[lo]));

        // Condition of target found
        if (arr[pos] == x)
            return pos;

        // If x is larger, x is in upper part
        if (arr[pos] < x)
            lo = pos + 1;

        // If x is smaller, x is in lower part
        else
            hi = pos - 1;
    }
    return -1;
}
//End Search sets

void PauseEnter() {
    printf("\nPress Enter to Continue..\n");
    getchar(); getchar();
}

void PrintDatas(int kounter, struct Datas Information[]){
    printf("%-2s | %-30s | %-10s | %-8s | %-50s\n", "No.", "Name", "Age",
      "Gender", "Float");
    printf("======================================================================\n");
    for (int i = 0; i < kounter; i++) {
        printf("%-2d | %-30s | %-8d | %-10s | %-50f\n", i, BioDatas[i].name, BioDatas[i].age, ((BioDatas[i].gender) ? "Male" : "Female"), BioDatas[i].fill);
    }
    printf("======================================================================\n");
}

void PrintBits(int n) {
    printf("======================================================================\n");
    printf("%-2d | %-30s | %-8d | %-10s | %-50f\n", n, BioDatas[n].name, BioDatas[n].age, ((BioDatas[n].gender) ? "Male" : "Female"), BioDatas[n].fill);
    printf("======================================================================\n");
}

int main() {
    FILE *fp;
    int select[10] = { -1,-1,-1,-1,-1 ,-1,-1,-1,-1,-1 };
    int kounter = 0;
    int flag = 0;
    bool flag_found = false;
    char confirm_rule[5];
    struct Datas InsertTemp;
    fp = fopen("datas.txt", "r");
    while (fscanf(fp, "%[^@]@%d@%d@%f\n", BioDatas[kounter].name, &BioDatas[kounter].age, &BioDatas[kounter].gender, &BioDatas[kounter].fill) != EOF) {
        printf("Loaded: %s, %d, %s, %f\n", BioDatas[kounter].name, BioDatas[kounter].age, ((BioDatas[kounter].gender)? "true" : "false"), BioDatas[kounter].fill);
        kounter++;
    };
    fclose(fp);

    printf("\nAll datas loaded!\n Enter to Start.\n");
    getchar();
    system("cls");
    do {
        system("cls");
        printf("Mockup Set, %d Bio Datas\n", kounter);
        printf("=========================================================\n");
        PrintDatas(kounter, BioDatas);
        printf("\n");
        printf("Select Section!\n");
        printf("\n");
        printf("1. Recursion, Struct, Search, Sort\n");
        printf("2. Insert, Update, Sort per Insert, Write save file\n");
        printf("\n");
        printf("9. Options\n");
        printf("0. Exit\n");
        printf("Choice > ");

        scanf("%d", &select[0]); fflush(stdin);

        switch (select[0]) {
        default:
            break;
        case 1: //section 1
            do {
                system("cls");
                printf("Section 1 (Recursion, Struct, Search, Sort)\n");
                printf("=====================\n");
                PrintDatas(kounter, BioDatas);
                printf("1. Search\n");
                printf("2. Sort\n");
                printf("\n");
                printf("\n");
                printf("0. Go back\n");
                printf("Choice > ");

                scanf("%d", &select[1]); fflush(stdin);

                switch (select[1]) {
                default:
                    break;
                case 1://Search
                    do {
                        printf("Which Name to Search?> ");
                        //scanf("%[^\n]%*c", InsertTemp.name); fflush(stdin);
                        scanf("%s", InsertTemp.name); fflush(stdin);
                    } while (strlen(InsertTemp.name) > 100);

                    flag = NamelinearSearch(BioDatas, kounter, InsertTemp.name);
                    if (flag != -1) {
                        printf("Data is Found!\n");
                        PrintBits(flag);
                    }
                    else printf("Data not Found!\n");

                    PauseEnter();
                    break;
                case 2://Sort
                    printf("Sort age\n");
                    AgebubbleSort(BioDatas, kounter);
                    printf("\nData Sorted by Age!\n");

                    PauseEnter();
                    break;
                }

            } while (select[1] != 0);
            select[1] = -1;
            break;
        case 2: //section 2
            do {
                system("cls");
                printf("Section 2 (Insert, Update, Sort per Insert, Write save file)\n");
                printf("======================\n");
                PrintDatas(kounter, BioDatas);
                printf("1. Add Data, then sort by name\n");
                printf("2. Edit Fill\n");
                printf("3. Delete Data\n");
                printf("\n");
                printf("0. Go back\n");
                printf("Choice > ");

                scanf("%d", &select[1]); fflush(stdin);

                switch (select[1]) {
                default:
                    break;
                case 1: //add data
                    printf("Register people\n\n");

                    do {
                        printf("Insert Name (100 char): ");
                        //scanf("%[^\n]%*c", InsertTemp.name); fflush(stdin);
                        scanf("%s", InsertTemp.name); fflush(stdin);
                    } while (strlen(InsertTemp.name) > 100);

                    flag = NamelinearSearch(BioDatas, kounter, InsertTemp.name);

                    if (flag == -1) {
                        printf("Insert Age: ");
                        scanf("%d", &InsertTemp.age); fflush(stdin);

                        do {
                            printf("Insert Gender (0 = female, 1 = male): ");
                            scanf("%d", &InsertTemp.gender); fflush(stdin);
                        } while (InsertTemp.gender < 0 || InsertTemp.gender > 1);

                        printf("Insert Fill: ");
                        scanf("%f", &InsertTemp.fill); fflush(stdin);

                        BioDatas[kounter] = InsertTemp;
                        kounter++;
                        printf("\nRegistered! Thx for joining!!\n");
                        PrintBits(kounter - 1);
                        PauseEnter();

                        printf("\nAutoSorting by name!\n");
                        NamebubbleSort(BioDatas, kounter);
                        printf("\nComplete\n");
                    }
                    else if (flag != -1) {
                        printf("\nData Already Exist!\n");
                        PrintBits(flag);
                        printf("\nPlease don't be immitator!\n");
                    }

                    PauseEnter();
                    break;
                case 2: //Edit Data fill
                    printf("Edit Fils\n");

                    do {
                        printf("Insert Name to edit fills (100 char): ");
                        //scanf("%[^\n]%*c", InsertTemp.name); fflush(stdin);
                        scanf("%s", InsertTemp.name); fflush(stdin);
                    } while (strlen(InsertTemp.name) > 100);

                    flag = NamelinearSearch(BioDatas, kounter, InsertTemp.name);

                    if (flag == -1) {
                        printf("Data is not found!\n");
                    }
                    else if (flag != -1) {
                        PrintBits(flag);

                        printf("Insert new Fill: ");
                        scanf("%f", &InsertTemp.fill); fflush(stdin);

                        BioDatas[flag].fill = InsertTemp.fill;
                        printf("Fill Updated!\n");
                        PrintBits(flag);
                    }

                    PauseEnter();
                    break;
                case 3: //delete data
                    do {
                        printf("Insert Name to Delete (100 char): ");
                        //scanf("%[^\n]%*c", InsertTemp.name); fflush(stdin);
                        scanf("%s", InsertTemp.name); fflush(stdin);
                    } while (strlen(InsertTemp.name) > 100);

                    flag = NamelinearSearch(BioDatas, kounter, InsertTemp.name);

                    if (flag == -1) {
                        printf("Data is not found! Maybe already deleted or not been here at all?\n");
                    }
                    else if (flag != -1) {
                        PrintBits(flag);
                        printf("\b\n");
                        printf("===========================================================================\n");
```

```cpp
                        printf("|       Warning! Are you Sure to delete this people above?       |\n");
                        printf("|       After Delete, no turning back!!!!!!!!!!!!!!!!!       |\n");
                        printf("===========================================================================\n");
                        printf("Confirm (YES / NO): ");

                        do {
                            scanf("%s", confirm_rule); fflush(stdin);
                        } while (strcmp(confirm_rule, "YES") != 0 && strcmp(confirm_rule, "NO") != 0);

                        if (strcmp(confirm_rule, "YES") == 0) {
                            for (int i = flag; i < kounter - 1; i++) {
                                BioDatas[i] = BioDatas[i + 1];
                            };
                            kounter--;

                            printf("Deleted. You such an evil!\n");
                        }
                        else if (strcmp(confirm_rule, "NO") == 0) {
                            printf("Delete Canceled. thank you for being kind.\n");
                        }

                    }

                    PauseEnter();
                    break;
                }

            } while (select[1] != 0);
            select[1] = -1;
            break;
        }

    } while (select[0] != 0);
    select[0] = -1;

    system("cls");
    printf("\nSaving data...\n");
    fp = fopen("datas.txt", "w");
    for (int i = 0; i < kounter; i++) {
        printf("Saving: %s, %d, %s, %f\n", BioDatas[i].name, BioDatas[i].age, ((BioDatas[i].gender) ? "true" : "false"), BioDatas[i].fill);
        fprintf(fp, "%s@%d@%d@%f\n", BioDatas[i].name, BioDatas[i].age, BioDatas[i].gender, BioDatas[i].fill);
    };
    fclose(fp);
    printf("\nSave Complete!\n");

    printf("\nProgram will Exit! Enter to Exit.");
    getchar();
    getchar();
    return 0;
}

```