

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<math.h>
4  #include<string.h>
5  #include<time.h>
6  #include<ctype.h>
7
8  /* GIANT NOTE:
9      Section 1:
10     - Recursive (Take a value #1 from array)
11     - Struct
12     - Search
13     - Sort
14
15     Section 2:
16     - Insert
17     - Update
18     - Sort every insert
19     - Write save file
20 */
21
22 /* Small Note
23 Sorting direction!:
24     - < Descending, high to low
25     - > Ascending, low to high
26     this is when the IF comparison.
27     The IF it means, was when comparing with the DATA in array themselves.
28 */
29
30 struct Datas {
31     char name[100];
32     int age;
33     bool gender; //male = true
34     float fill;
35 } BioDatas[1000];
36
37 //Base functions
38 void swapy(int *xp, int *yp)
39 {
40     int temp = *xp;
41     *xp = *yp;
42     *yp = temp;
43 }
44
45 void swapStruct(struct Datas *InfoA, struct Datas *InfoB) {
46     struct Datas temps = *InfoA;
47     *InfoA = *InfoB;
48     *InfoB = temps;
49 }
50 // end base functions
51
52 //Install! Sort Algorithms. collected by Geeks for geeks
53 //Begin Sort sets
54 // A function to implement bubble sort
55 void bubbleSort(int arr[], int n)
56 {
```

```
57     int i, j;
58     for (i = 0; i < n - 1; i++)
59
60         // Last i elements are already in place
61         for (j = 0; j < n - i - 1; j++)
62             if (arr[j] > arr[j + 1])
63                 swapy(&arr[j], &arr[j + 1]);
64 }
65
66 void AgebubbleSort(struct Datas informations[], int n) {
67     for (int i = 0; i < n; i++) {
68         for (int j = 0; j < n - i - 1; j++) {
69             if (informations[j].age < informations[j + 1].age) { //Descending ↗
70                 <
71                 swapStruct(&informations[j], &informations[j + 1]);
72             }
73         }
74     }
75
76 void NamebubbleSort(struct Datas informations[], int n) {
77     for (int i = 0; i < n; i++) {
78         for (int j = 0; j < n - i - 1; j++) {
79             if (strcmp(informations[j].name, informations[j + 1].name) > 0) ↗
80                 { //Ascending >
81                     swapStruct(&informations[j], &informations[j + 1]);
82                 }
83         }
84     }
85
86 void selectionSort(int arr[], int n)
87 {
88     int i, j, min_idx;
89
90     // One by one move boundary of unsorted subarray
91     for (i = 0; i < n - 1; i++)
92     {
93         // Find the minimum element in unsorted array
94         min_idx = i;
95         for (j = i + 1; j < n; j++)
96             if (arr[j] < arr[min_idx])
97                 min_idx = j;
98
99         // Swap the found minimum element with the first element
100        swapy(&arr[min_idx], &arr[i]);
101    }
102 }
103
104 void insertionSort(int arr[], int n)
105 {
106     int i, key, j;
107     for (i = 1; i < n; i++)
108     {
109         key = arr[i];
110         j = i - 1;
```

```
111
112     /* Move elements of arr[0..i-1], that are
113     greater than key, to one position ahead
114     of their current position */
115     while (j >= 0 && arr[j] > key)
116     {
117         arr[j + 1] = arr[j];
118         j = j - 1;
119     }
120     arr[j + 1] = key;
121 }
122 }
123
124 //Merge Sort begin
125 // Merges two subarrays of arr[].
126 // First subarray is arr[l..m]
127 // Second subarray is arr[m+1..r]
128 //void merge(int arr[], int l, int m, int r)
129 //{
130 //    int i, j, k;
131 //    int n1 = m - l + 1;
132 //    int n2 = r - m;
133 //
134 //    /* create temp arrays */
135 //    int L[n1], R[n2];
136 //
137 //    /* Copy data to temp arrays L[] and R[] */
138 //    for (i = 0; i < n1; i++)
139 //        L[i] = arr[l + i];
140 //    for (j = 0; j < n2; j++)
141 //        R[j] = arr[m + 1 + j];
142 //
143 //    /* Merge the temp arrays back into arr[l..r]*/
144 //    i = 0; // Initial index of first subarray
145 //    j = 0; // Initial index of second subarray
146 //    k = l; // Initial index of merged subarray
147 //    while (i < n1 && j < n2)
148 //    {
149 //        if (L[i] <= R[j])
150 //        {
151 //            arr[k] = L[i];
152 //            i++;
153 //        }
154 //        else
155 //        {
156 //            arr[k] = R[j];
157 //            j++;
158 //        }
159 //        k++;
160 //    }
161 //
162 //    /* Copy the remaining elements of L[], if there
163 //    are any */
164 //    while (i < n1)
165 //    {
166 //        arr[k] = L[i];
```

```
167 //      i++;
168 //      k++;
169 //  }
170 //
171 //  /* Copy the remaining elements of R[], if there
172 //  are any */
173 //  while (j < n2)
174 //  {
175 //      arr[k] = R[j];
176 //      j++;
177 //      k++;
178 //  }
179 //}
180 //
181 ///* l is for left index and r is right index of the
182 //sub-array of arr to be sorted */
183 //void mergeSort(int arr[], int l, int r)
184 //{
185 //  if (l < r)
186 //  {
187 //      // Same as (l+r)/2, but avoids overflow for
188 //      // large l and h
189 //      int m = l + (r - l) / 2;
190 //
191 //      // Sort first and second halves
192 //      mergeSort(arr, l, m);
193 //      mergeSort(arr, m + 1, r);
194 //
195 //      merge(arr, l, m, r);
196 //  }
197 //}
198 //Merge Sort Ends
199
200 //Quick Sort Begin
201 /* This function takes last element as pivot, places
202 the pivot element at its correct position in sorted
203 array, and places all smaller (smaller than pivot)
204 to left of pivot and all greater elements to right
205 of pivot */
206 int partition(int arr[], int low, int high)
207 {
208     int pivot = arr[high];    // pivot
209     int i = (low - 1);    // Index of smaller element
210
211     for (int j = low; j <= high - 1; j++)
212     {
213         // If current element is smaller than or
214         // equal to pivot
215         if (arr[j] <= pivot)
216         {
217             i++;    // increment index of smaller element
218             swapy(&arr[i], &arr[j]);
219         }
220     }
221     swapy(&arr[i + 1], &arr[high]);
222     return (i + 1);
```

```
223 }
224
225 /* The main function that implements QuickSort
226 arr[] --> Array to be sorted,
227 low --> Starting index,
228 high --> Ending index */
229 void quickSort(int arr[], int low, int high)
230 {
231     if (low < high)
232     {
233         /* pi is partitioning index, arr[p] is now
234         at right place */
235         int pi = partition(arr, low, high);
236
237         // Separately sort elements before
238         // partition and after partition
239         quickSort(arr, low, pi - 1);
240         quickSort(arr, pi + 1, high);
241     }
242 }
243 //Quick Sort Ended
244
245 //End Sort sets
246
247 //Install! Search Algorithms. collected by Geeks for geeks
248 //Begin Search sets
249 // Linearly search x in arr[]. If x is present then return its
250 // location, otherwise return -1
251 int linearSearch(int arr[], int n, int x)
252 {
253     int i;
254     for (i = 0; i < n; i++)
255         if (arr[i] == x)
256             return i;
257     return -1;
258 }
259
260 int NamelinearSearch(struct Datas informations[], int n, char name[]) {
261     for (int i = 0; i < n; i++) {
262         if (strcmp(informations[i].name, name) == 0) {
263             return i;
264         }
265     }
266     return -1;
267 }
268
269 int binarySearch(int arr[], int l, int r, int x)
270 {
271     if (r >= l)
272     {
273         int mid = l + (r - l) / 2;
274
275         // If the element is present at the middle
276         // itself
277         if (arr[mid] == x)
278             return mid;
```

```
279
280     // If element is smaller than mid, then
281     // it can only be present in left subarray
282     if (arr[mid] > x)
283         return binarySearch(arr, l, mid - 1, x);
284
285     // Else the element can only be present
286     // in right subarray
287     return binarySearch(arr, mid + 1, r, x);
288 }
289
290 // We reach here when element is not
291 // present in array
292 return -1;
293 }
294
295 // If x is present in arr[0..n-1], then returns
296 // index of it, else returns -1.
297 int interpolationSearch(int arr[], int n, int x)
298 {
299     // Find indexes of two corners
300     int lo = 0, hi = (n - 1);
301
302     // Since array is sorted, an element present
303     // in array must be in range defined by corner
304     while (lo <= hi && x >= arr[lo] && x <= arr[hi])
305     {
306         // Probing the position with keeping
307         // uniform distribution in mind.
308         int pos = lo + (((double)(hi - lo) /
309             (arr[hi] - arr[lo]))*(x - arr[lo]));
310
311         // Condition of target found
312         if (arr[pos] == x)
313             return pos;
314
315         // If x is larger, x is in upper part
316         if (arr[pos] < x)
317             lo = pos + 1;
318
319         // If x is smaller, x is in lower part
320         else
321             hi = pos - 1;
322     }
323     return -1;
324 }
325 //End Search sets
326
327 void PauseEnter() {
328     printf("\nPress Enter to Continue..\n");
329     getchar(); getchar();
330 }
331
332 void PrintDatas(int kounter, struct Datas Information[]){
333     printf("%-2s | %-30s | %-10s | %-8s | %-50s\n", "No.", "Name", "Age",
        "Gender", "Float");
```

```

...8\JOELwindows7_Mockup_training_FinalExam2018\Source.cpp 7
334     printf
335     (
336         "\n");
337     for (int i = 0; i < kounter; i++) {
338         printf("%-2d | %-30s | %-8d | %-10s | %-50f\n", i, BioDatas[i].name,
339             BioDatas[i].age, ((BioDatas[i].gender) ? "Male" : "Female"),
340             BioDatas[i].fill);
341     }
342     printf
343     (
344         "\n");
345 }
346
347 void PrintBits(int n) {
348     printf
349     (
350         "\n");
351     printf("%-2d | %-30s | %-8d | %-10s | %-50f\n", n, BioDatas[n].name,
352         BioDatas[n].age, ((BioDatas[n].gender) ? "Male" : "Female"), BioDatas
353         [n].fill);
354     printf
355     (
356         "\n");
357 }
358
359 int main() {
360     FILE *fp;
361     int select[10] = { -1,-1,-1,-1,-1 , -1,-1,-1,-1,-1 };
362     int kounter = 0;
363     int flag = 0;
364     bool flag_found = false;
365     char confirm_rule[5];
366     struct Datas InsertTemp;
367     fp = fopen("datas.txt", "r");
368     while (fscanf(fp, "%[^@]@%d@%d@%f\n", BioDatas[kounter].name, &BioDatas
369         [kounter].age, &BioDatas[kounter].gender, &BioDatas[kounter].fill) !=
370         EOF) {
371         printf("Loaded: %s, %d, %s, %f\n", BioDatas[kounter].name, BioDatas
372             [kounter].age, ((BioDatas[kounter].gender)? "true" : "false"),
373             BioDatas[kounter].fill);
374         kounter++;
375     };
376     fclose(fp);
377
378     printf("\nAll datas loaded!\n Enter to Start.\n");
379     getchar();
380     system("cls");
381     do {
382         system("cls");
383         printf("Mockup Set, %d Bio Datas\n", kounter);
384         printf("=====\n");
385         PrintDatas(kounter, BioDatas);
386         printf("\n");
387         printf("Select Section!\n");
388         printf("\n");
389         printf("1. Recursion, Struct, Search, Sort\n");

```

```
374     printf("2. Insert, Update, Sort per Insert, Write save file\n");
375     printf("\n");
376     printf("9. Options\n");
377     printf("0. Exit\n");
378     printf("Choice > ");
379
380     scanf("%d", &select[0]); fflush(stdin);
381
382     switch (select[0]) {
383     default:
384         break;
385     case 1: //section 1
386         do {
387             system("cls");
388             printf("Section 1 (Recursion, Struct, Search, Sort)\n");
389             printf("=====\n");
390             PrintDatas(kounter, BioDatas);
391             printf("1. Search\n");
392             printf("2. Sort\n");
393             printf("\n");
394             printf("\n");
395             printf("0. Go back\n");
396             printf("Choice > ");
397
398             scanf("%d", &select[1]); fflush(stdin);
399
400             switch (select[1]) {
401             default:
402                 break;
403             case 1://Search
404                 do {
405                     printf("Which Name to Search?> ");
406                     //scanf("%[^\n]*c", InsertTemp.name); fflush(stdin);
407                     scanf("%s", InsertTemp.name); fflush(stdin);
408                 } while (strlen(InsertTemp.name) > 100);
409
410                 flag = NamelinearSearch(BioDatas, kounter,
411                                     InsertTemp.name);
412                 if (flag != -1) {
413                     printf("Data is Found!\n");
414                     PrintBits(flag);
415                 }
416                 else printf("Data not Found!\n");
417
418                 PauseEnter();
419                 break;
420             case 2://Sort
421                 printf("Sort age\n");
422                 AgebubbleSort(BioDatas, kounter);
423                 printf("\nData Sorted by Age!\n");
424
425                 PauseEnter();
426                 break;
427             }
428         } while (select[1] != 0);
```



```
429         select[1] = -1;
430         break;
431     case 2: //section 2
432         do {
433             system("cls");
434             printf("Section 2 (Insert, Update, Sort per Insert, Write save
435                 file)\n");
436             printf("=====\n");
437             PrintDatas(kounter, BioDatas);
438             printf("1. Add Data, then sort by name\n");
439             printf("2. Edit Fill\n");
440             printf("3. Delete Data\n");
441             printf("\n");
442             printf("0. Go back\n");
443             printf("Choice > ");
444
445             scanf("%d", &select[1]); fflush(stdin);
446
447             switch (select[1]) {
448             default:
449                 break;
450             case 1: //add data
451                 printf("Register people\n\n");
452                 do {
453                     printf("Insert Name (100 char): ");
454                     //scanf("%[^\n]*c", InsertTemp.name); fflush(stdin);
455                     scanf("%s", InsertTemp.name); fflush(stdin);
456                 } while (strlen(InsertTemp.name) > 100);
457
458                 flag = NamelinearSearch(BioDatas, kounter,
459                     InsertTemp.name);
460
461                 if (flag == -1) {
462                     printf("Insert Age: ");
463                     scanf("%d", &InsertTemp.age); fflush(stdin);
464
465                     do {
466                         printf("Insert Gender (0 = female, 1 = male): ");
467                         scanf("%d", &InsertTemp.gender); fflush(stdin);
468                     } while (InsertTemp.gender < 0 || InsertTemp.gender >
469                         1);
470
471                     printf("Insert Fill: ");
472                     scanf("%f", &InsertTemp.fill); fflush(stdin);
473
474                     BioDatas[kounter] = InsertTemp;
475                     kounter++;
476                     printf("\nRegistered! Thx for joining!!\n");
477                     PrintBits(kounter - 1);
478                     PauseEnter();
479
480                     printf("\nAutoSorting by name!\n");
481                     NamebubbleSort(BioDatas, kounter);
482                     printf("\nComplete\n");
483                 }
484             }
```

```

482         else if (flag != -1) {
483             printf("\nData Already Exist!\n");
484             PrintBits(flag);
485             printf("\nPlease don't be immitator!\n");
486         }
487
488         PauseEnter();
489         break;
490     case 2: //Edit Data fill
491         printf("Edit Fils\n");
492
493         do {
494             printf("Insert Name to edit fills (100 char): ");
495             //scanf("%[^\\n]*c", InsertTemp.name); fflush(stdin);
496             scanf("%s", InsertTemp.name); fflush(stdin);
497         } while (strlen(InsertTemp.name) > 100);
498
499         flag = NamelinearSearch(BioDatas, kounter,
                                InsertTemp.name);
500
501         if (flag == -1) {
502             printf("Data is not found!\n");
503         }
504         else if (flag != -1) {
505             PrintBits(flag);
506
507             printf("Insert new Fill: ");
508             scanf("%f", &InsertTemp.fill); fflush(stdin);
509
510             BioDatas[flag].fill = InsertTemp.fill;
511             printf("Fill Updated!\n");
512             PrintBits(flag);
513         }
514
515         PauseEnter();
516         break;
517     case 3: //delete data
518         do {
519             printf("Insert Name to Delete (100 char): ");
520             //scanf("%[^\\n]*c", InsertTemp.name); fflush(stdin);
521             scanf("%s", InsertTemp.name); fflush(stdin);
522         } while (strlen(InsertTemp.name) > 100);
523
524         flag = NamelinearSearch(BioDatas, kounter,
                                InsertTemp.name);
525
526         if (flag == -1) {
527             printf("Data is not found! Maybe already deleted or
not been here at all?\n");
528         }
529         else if (flag != -1) {
530             PrintBits(flag);
531             printf("\b\n");
532             printf
("=====\n");

```

```

...8\JOELwindows7_Mockup_training_FinalExam2018\Source.cpp 11
533         printf("|      Warning! Are you Sure to delete this  7
           people above?      |\n");
534         printf("|      After Delete, no turning          7
           back!!!!!!!!!!!!!!      |\n");
535         printf                                          7
           ("===== 7
           =====\n");
536         printf("Confirm (YES / NO): ");
537
538         do {
539             scanf("%s", confirm_rule); fflush(stdin);
540         } while (strcmp(confirm_rule, "YES") != 0 && strcmp 7
           (confirm_rule, "NO") != 0);
541
542         if (strcmp(confirm_rule, "YES") == 0) {
543             for (int i = flag; i < kounter - 1; i++) {
544                 BioDatas[i] = BioDatas[i + 1];
545             };
546             kounter--;
547
548             printf("Deleted. You such an evil!\n");
549         }
550         else if (strcmp(confirm_rule, "NO") == 0) {
551             printf("Delete Canceled. thank you for being kind. 7
           \n");
552         }
553
554     }
555
556     PauseEnter();
557     break;
558 }
559
560     } while (select[1] != 0);
561     select[1] = -1;
562     break;
563 }
564
565 } while (select[0] != 0);
566 select[0] = -1;
567
568 system("cls");
569 printf("\nSaving data...\n");
570 fp = fopen("datas.txt", "w");
571 for (int i = 0; i < kounter; i++) {
572     printf("Saving: %s, %d, %s, %f\n", BioDatas[i].name, BioDatas[i].age, 7
           ((BioDatas[i].gender) ? "true" : "false"), BioDatas[i].fill);
573     fprintf(fp, "%s@%d@%d@%f\n", BioDatas[i].name, BioDatas[i].age, 7
           BioDatas[i].gender, BioDatas[i].fill);
574 };
575 fclose(fp);
576 printf("\nSave Complete!\n");
577
578 printf("\nProgram will Exit! Enter to Exit.");
579 getchar();
580 getchar();

```

---

```
581     return 0;  
582 }  
583
```