

```
1  /*
2  Course: Hardware oriented programming
3  Assignment: 10
4  Student: Mads Richardt
5  Student ID: s224948
6  */
7
8  #include <stdlib.h>
9  #include <stdio.h>
10 #include <string.h>
11 #define MAX_NAME_LENGTH 50
12 #define MAX_ADDRESS_LENGTH 100
13 #define MAX_DATA_BASE_LENGTH 100
14
15 // Declaration of person struct
16 typedef struct person
17 {
18     char firstName[MAX_NAME_LENGTH];
19     char lastName[MAX_NAME_LENGTH];
20     unsigned int age;
21     char address[MAX_ADDRESS_LENGTH];
22     size_t phoneNumber;
23     struct person *nextPtr;
24 } Person;
25
26 // Function declarations
27 int addPersonToCsvFile(char *filename);
28 Person *stringToPersonPointer(char *string);
29 Person *scvFileToLinkedList(char *fileName, size_t *personCount);
30 int sortLinkedPersonList(Person **startNode, size_t personCount);
31 Person *swapNodes(Person *node1Ptr, Person *node2Ptr);
32 Person *getMiddleNode(Person *headNode, size_t length);
33 Person *linkedListBinarySearch(Person *headNode, size_t value, size_t length);
34 int viewPerson(Person *person);
35
36 int main()
37 {
38
39     char fileName[MAX_NAME_LENGTH] = "database";
40     size_t personCount = 0;
41     int controlVar = 0;
42     Person *headNode = NULL;
43     int sortedFlag = 0;
44
45     puts("*****");
46     puts("Welcome to Assignment 9");
47     puts("*****");
48
49     // Get file name from user
50     printf("\nEnter data base name: ");
51     scanf("%s", fileName);
52
53     // Load SCV file into linked list.
54     headNode = scvFileToLinkedList(fileName, &personCount);
55
56     // Sort linked list.
57     sortLinkedPersonList(&headNode, personCount);
58     sortedFlag = 1;
59
60
61     // Check for format error
```

```

62     if (personCount > 0 && headNode == NULL)
63     {
64         printf("Format error on line %lu of csv file %s\n", personCount,
fileName);
65         printf("Program closing...");
66         exit(0);
67     }
68     // Check if file exists
69     if (personCount == 0 && headNode == NULL)
70     {
71         printf("File could not be opened.");
72         printf("Program closing...");
73         exit(0);
74     }
75
76     do
77     {
78         printf("\n1: Add person to %s.\n2: Search %s on Phone Number.\n3:
Close Program.\nPlease choose option: ", fileName, fileName);
79         scanf("%1d", &controlVar);
80         puts("");
81
82         switch (controlVar)
83         {
84             case 1:
85             {
86                 addPersonToCsvFile(fileName);
87                 sortedFlag = 0;
88                 break;
89             }
90             case 2:
91             {
92                 size_t searchNumber;
93
94                 // Get phone number
95                 printf("Enter phone number: ");
96                 scanf("%lu", &searchNumber);
97
98                 // Sort if not sorted
99                 if (sortedFlag == 0)
100                 {
101                     personCount = 0;
102
103                     // Load SCV file into linked list.
104                     headNode = scvFileToLinkedList(fileName,
&personCount);
105
106                     // Sort linked list
107                     sortLinkedPersonList(&headNode,
personCount);
108
109                     // set sorted flag
110                     sortedFlag = 1;
111                 }
112
113                 Person *result = linkedListBinarySearch(headNode,
searchNumber, personCount);
114
115                 if (result == NULL)
116                 {
117                     puts("No match found.");
118                 }

```

```

119         else {
120             puts("\nMatch found:");
121             viewPerson(result);
122         }
123
124         break;
125     }
126     default:
127         break;
128 }
129
130 } while (controlVar != 3);
131
132 // Close program.
133 puts("Program closing...");
134 return 0;
135 }
136
137 Person *LinkedListBinarySearch(Person *headNode, size_t value, size_t length)
138 {
139     Person *startNode = headNode;
140
141     do
142     {
143         // Get middle node.
144         Person *middleNode = getMiddleNode(startNode, length);
145
146         // Return NULL if middle is empty
147         if (middleNode == NULL)
148         {
149             return NULL;
150         }
151
152         // If middleNode contains value, return middleNode
153         if (middleNode->phoneNumber == value)
154         {
155             return middleNode;
156         }
157
158         // If value larger than middleNode->phoneNumber
159         if (middleNode->phoneNumber < value)
160         {
161             startNode = middleNode->nextPtr;
162             length = length/2;
163         }
164         else
165         {
166             length = length/2;
167         }
168     } while (1);
169
170     return NULL;
171 }
172
173 int addPersonToCsvFile(char *filename)
174 {
175     // Open file in append mode
176     FILE *fPtr = fopen(filename, "a+");
177
178     // Creat Person struct
179     Person person;

```

```

180
181 // Get first name from user.
182 printf("Enter First Name: ");
183 scanf("%s", person.firstName);
184
185 // Get last name from user.
186 printf("Enter Last Name: ");
187 scanf("%s", person.lastName);
188
189 // Get age from user.
190 printf("Enter Age: ");
191 scanf("%u", &person.age);
192
193 // Get address from user.
194 printf("Enter Address: ");
195 scanf("%*[\n]%^[\n]", person.address);
196
197 // Get phone number from user.
198 printf("Enter Phone Number: ");
199 scanf("%lu", &person.phoneNumber);
200
201 // Append person to SCV file.
202 fprintf(fPtr, "\n%s, %s, %u, %s, %lu", person.firstName, person.lastName,
person.age, person.address, person.phoneNumber);
203 fclose(fPtr);
204
205 return 1;
206 }
207
208 Person *stringToPersonPointer(char *string)
209 {
210 // Declare person pointer.
211 Person *person;
212
213 // Initialize format string.
214 char *formatString = "%[^,]*[, ]%[^,]*[, ]%u*[, ]%[^,]*[, ]%zu";
215
216 // Allocate space for Person struct in heap.
217 person = (Person *)malloc(sizeof(Person));
218
219 // Scan string into person members.
220 int scanned = sscanf(string, formatString, person->firstName, person-
>lastName, &person->age, person->address, &person->phoneNumber);
221
222 // Set nextPtr to NULL.
223 person->nextPtr = NULL;
224
225 // Set person to NULL if all members were not scanned correctly.
226 if (scanned != 5)
227 {
228     person = NULL;
229 }
230
231 // return Person pointer.
232 return person;
233 }
234
235 Person *scvFileToLinkedList(char *fileName, size_t *personCount)
236 {
237 // Declare node pointers.
238 Person *startNode, *tempNode, *currentNode;
239

```

```

240 // Open file
241 FILE *fPtr = fopen(fileName, "r+");
242
243 // If file does not exists return fPtr.
244 if (fPtr == NULL)
245 {
246     return (Person *)fPtr;
247 }
248
249 // Declare getline() buffer pointer.
250 char *line = NULL;
251
252 // Declare getline() buffer size.
253 size_t len = 0;
254
255 // Scan file line by line.
256 while ((getline(&line, &len, fPtr)) != -1)
257 {
258     if (*personCount == 0)
259     {
260         // Increment personCount
261         *personCount = *personCount + (size_t)1;
262
263         // Initialize startNode.
264
265         startNode = stringToPersonPointer(line);
266
267         // Return NULL if stringToPersonPointer() did not scan all
members correctly.
268         if (startNode == NULL)
269         {
270             // Free line buffer
271             free(line);
272             // Close file
273             fclose(fPtr);
274             return startNode;
275         }
276
277         tempNode = startNode;
278     }
279     else
280     {
281         // Increment personCount.
282         *personCount = *personCount + (size_t)1;
283
284         // Initialize currentNode.
285         currentNode = stringToPersonPointer(line);
286
287         // Return NULL if stringToPersonPointer() did not scan all
members correctly.
288         if (currentNode == NULL)
289         {
290             // Free line buffer
291             free(line);
292             // Close file
293             fclose(fPtr);
294             return currentNode;
295         }
296
297         tempNode->nextPtr = currentNode;
298         tempNode = currentNode;

```

```
299     }
300 }
301
302 // Free line buffer
303 free(line);
304 // Close file
305 fclose(fPtr);
306
307 return startNode;
308 }
309
310 int sortLinkedList(Person **startNode, size_t personCount)
311 {
312     Person **tempNode;
313     size_t swapped;
314     Person *p1, *p2;
315
316     // Outer loop
317     for (size_t i = 0; i <= personCount; i++)
318     {
319         tempNode = startNode;
320         swapped = 0;
321
322         // Inner loop
323         for (size_t j = 0; j < personCount - i - 1; j++)
324         {
325             p1 = *tempNode;
326             p2 = p1->nextPtr;
327
328             if (p1->phoneNumber > p2->phoneNumber)
329             {
330                 *tempNode = swapNodes(p1, p2);
331                 swapped = 1;
332             }
333
334             tempNode = &(*tempNode)->nextPtr;
335         }
336         if (swapped == 0)
337         {
338             break;
339         }
340     }
341     return 1;
342 }
343
344 Person *swapNodes(Person *node1Ptr, Person *node2Ptr)
345 {
346     Person *temp = node2Ptr->nextPtr;
347     node2Ptr->nextPtr = node1Ptr;
348     node1Ptr->nextPtr = temp;
349
350     return node2Ptr;
351 }
352
353 Person *getMiddleNode(Person *headNode, size_t length)
354 {
355     Person *middleNode = headNode;
356     // find middle node
357     for (size_t i = 1; i < length / 2; i++)
358     {
359         middleNode = middleNode->nextPtr;
360     }
361     return middleNode;
362 }
```

```
360         middleNode = middleNode->nextPtr;
361     }
362     return middleNode;
363 }
364
365 int viewPerson(Person *person)
366 {
367     printf("Name: %s %s\n", person->firstName, person->lastName);
368     printf("Age: %u\n", person->age);
369     printf("Address: %s\n", person->address);
370     printf("Phone number: %lu\n", person->phoneNumber);
371     return 1;
372 }
```