**ПИК 1**

**КУРСОВА РАБОТА**

**ТУ-София**

1. **Блок схема**



1. **Код**
2. #include<stdio.h>
3. #include<stdlib.h>
4. #include<malloc.h>
5. #include<time.h>
6. int InitializeArraySize();
7. void FillArrayWithRand(int\* array, int size);
8. int GetRandomNumber(int min\_number, int max\_number);
9. void PrintArray(int array[], int size);
10. int GetMinIndex(int array[], int size);
11. void CopyArrays(int\* array1, int\* array2, int sizeOfSmallerArray);
12. void quickSort(int array[], int low, int high);
13. int partition(int array[], int low, int high);
14. void swap(int \*one, int\* two);
15. int GetRepNumsCount(int arr[], int size);
16. void PrintAllRepNums(int array[], int size);
17. void ArrayWorkWithRandomNumbers();
18. void ArrayWorkWithManualNumbers();
19. void SaveArray(int array[], int size);
20. void EditArray(int\* array, int size);
21. int main()
22. {
23. srand(time(0));
24. int choice = 0;
25. main\_menu:
26. printf("\nWhat would you want to do?\n1-Array work\n2-Exit\n");
27. scanf("%d", &choice);
28. switch (choice)
29. {
30. case 1:
31. {
32. secondmenu:;
33. int secondChoice = 0;
34. printf("\nWhat would you want to do?\n1-Create an array \n2-Open an array\n3-Back\n");
35. scanf("%d", &secondChoice);
36. switch (secondChoice)
37. {
38. case 1:
39. //Create an aray:
40. randomOrManualChoice:;
41. int randomOrManual = 0;
42. printf("\nChoose:\n1-Random Numbers\n2-Manual mode\n3-Back\n");
43. scanf("%d", &randomOrManual);
44. switch (randomOrManual)
45. {
46. case 1:
47. //random numbers array
48. ArrayWorkWithRandomNumbers();
49. goto main\_menu;
50. break;
51. case 2:
52. ArrayWorkWithManualNumbers();
53. break;
54. case 3:
55. //go to back(to second menu)
56. goto secondmenu;
57. break;
58. default:
59. printf("Invalid choice!\n\n");
60. goto randomOrManualChoice;
61. break;
62. }
63. break;
64. case 2:
65. //TODO Opening an array from file
66. break;
67. case 3:
68. //Back (main menu)
69. goto main\_menu;
70. break;
71. default:
72. printf("Invalid choice!\n\n");
73. goto secondmenu;
74. break;
75. }
76. break;
77. }
78. case 2:
79. break;
80. default:
81. {
82. printf("Invalid choice!\n\n");
83. goto main\_menu;
84. break;
85. }
86. }
87. return 0;
88. }
89. int InitializeArraySize()
90. {
91. //User enters size for an array
92. unsigned int size = 0;
93. printf("How big should the InitialArray be? (≤100)\n");
94. here:
95. scanf("%d, size", &size);
96. if(size < 1 || size > 100)
97. {
98. printf("Invalid size! Choose a number between 1 and 100\n");
99. goto here;
100. }
101. return size;
102. }
103. void FillArrayWithRand(int\* array, int size)
104. {
105. //Fills array with random numbers
106. for (int i = 0; i < size; i++)
107. {
108. array[i] = GetRandomNumber(0, 100);
109. }
110. }
111. int GetRandomNumber(int min\_number, int max\_number)
112. {
113. //Generates a random number
114. int number = (rand() % (max\_number + 1 - min\_number )) + min\_number;
115. return number;
116. }
117. void PrintArray(int array[], int size)
118. {
119. //Prints array
120. for (int i = 0; i < size; i++)
121. {
122. printf("[%d] = %d\n", i, array[i]);
123. }
124. }
125. int GetMinIndex(int array[], int size)
126. {
127. //Returns the index of the min number in array
128. int minnum = array[0];
129. int index = 0;
130. for (int i = 1; i < size; i++)
131. {
132. if (array[i] < minnum)
133. {
134. minnum = array[i];
135. index = i;
136. }
137. }
138. return index;
139. }
140. void CopyArrays(int\* array1, int\* array2, int sizeOfSmallerArray)
141. {
142. //Copies the elements from array1 to array2 (array2 MUST BE THE SAME SIZE OF SMALLER THAN array1)
143. for (int i = 0; i < sizeOfSmallerArray; i++)
144. {
145. array2[i] = array1[i];
146. }
147. }
148. void quickSort(int array[], int low, int high)
149. {
150. //Self-explanatory
151. if (low < high)
152. {
153. int pi = partition(array, low, high);
154. quickSort(array, low, pi-1);
155. quickSort(array, pi+1, high);
156. }
157. }
158. int partition(int array[], int low, int high)
159. {
160. //Teruns the pivot for quickSort
161. int pivot = array[high];
162. int i = low - 1;
163. for (int j = low; j <= high - 1; j++)
164. {
165. if (array[j] < pivot)
166. {
167. i++;
168. swap(&array[i], &array[j]);
169. }
170. }
171. swap(&array[i+1], &array[high]);
172. return (i+1);
173. }
174. void swap(int \*one, int\* two)
175. {
176. //swaps the values of 'one' and 'two'
177. int temp = \*one;
178. \*one = \*two;
179. \*two = temp;
180. }
181. int GetRepNumsCount(int arr[], int size)
182. {
183. //Returns the Count of the numbers that are repeated in array
184. //NOTE: the function doesn't return how many times the number has been returned
185. int count = 0, j;
186. for (int i = 0; i < size - 1; i++)
187. {
188. j = 1;
189. while(arr[i] == arr[i+1])
190. {
191. i++;
192. j++;
193. }
194. if (j != 1)
195. {
196. count++;
197. }
198. }
199. return count;
200. }
201. void PrintAllRepNums(int array[], int size)
202. {
203. //Prints every number that is repeated in array AND how many times it has been repeated
204. int RepNumsCount = GetRepNumsCount(array,size);
205. printf("Repeated Numbers: %d\n", RepNumsCount);
206. if (RepNumsCount != 0)
207. {
208. int\*\* RepesatingNumbers = (int\*\*)malloc(RepNumsCount \* sizeof(int));
209. for (int i = 0; i < RepNumsCount; i++){RepesatingNumbers[i] = (int\*)malloc(2\*sizeof(int));}
211. int j = 0;
212. for (int i = 0; i < RepNumsCount; i++)
213. {
214. int count = 1;
215. while (array[j] != array[j+1])
216. {
217. j++;
218. }
219. while (array[j] == array[j+1])
220. {
221. j++;
222. count++;
223. }
224. RepesatingNumbers[i][0] = array[j];
225. RepesatingNumbers[i][1] = count;
226. printf("%d is repeated %d times\n",RepesatingNumbers[i][0], RepesatingNumbers[i][1]);
227. }
228. }
229. }
230. void ArrayWorkWithRandomNumbers()
231. {
232. int InitialArraySize, SecondArraySize, EditChoice;
233. InitialArraySize = InitializeArraySize();
234. int\* InitialArray = (int\*) malloc(InitialArraySize \* sizeof(int));
235. FillArrayWithRand(InitialArray, InitialArraySize);
236. PrintArray(InitialArray, InitialArraySize);
237. printf("Before continuing, would you want to edit the array?\n1-Yes\n2-No\n");
238. editchoice\_manual:
239. scanf("%d", & EditChoice);
240. if (EditChoice == 1)
241. {
242. EditArray(InitialArray, InitialArraySize);
243. }
244. else if (EditChoice == 2)
245. {
246. }
247. else
248. {
249. printf("Invalid choice!\nEnter 1 or 2\n");
250. goto editchoice\_manual;
251. }
252. PrintArray(InitialArray, InitialArraySize);
253. SecondArraySize = GetMinIndex(InitialArray, InitialArraySize);
254. printf("Min index is %d and InitialArray[%d] = %d\n", SecondArraySize, SecondArraySize, InitialArray[SecondArraySize]);
255. int\* SecondArray = (int\*) malloc(SecondArraySize \* sizeof(int));
256. CopyArrays(InitialArray, SecondArray, SecondArraySize);
257. printf("---------\nNew Array:\n");
258. PrintArray(SecondArray, SecondArraySize);
259. printf("---------\nSorted New Array:\n");
260. quickSort(SecondArray, 0, SecondArraySize - 1);
261. PrintArray(SecondArray,SecondArraySize);
262. printf("----------\n");
263. PrintAllRepNums(SecondArray,SecondArraySize);
264. int save = 0;
265. save\_option\_auto:
266. printf("\nWould you like to Save the array?\n1-Yes\n2-No(Exit to Main Menu)\n");
267. scanf("%d", &save);
268. if (save == 1)
269. {
270. SaveArray(SecondArray,SecondArraySize);
271. }
272. else if (save == 2)
273. {
274. exit;
275. }
276. else
277. {
278. printf("Invalid choice!\n");
279. goto save\_option\_auto;
280. }
281. }
282. void ArrayWorkWithManualNumbers()
283. {
284. int InitialArraySize, SecondArraySize, EditChoice;
285. InitialArraySize = InitializeArraySize();
286. int\* InitialArray = (int\*) malloc(InitialArraySize \* sizeof(int));
287. for (int i = 0; i < InitialArraySize; i++)
288. {
289. printf("Enter number[%d] :", i);
290. scanf("%d", &InitialArray[i]);
291. }
292. PrintArray(InitialArray, InitialArraySize);
293. printf("Before continuing, would you want to edit the array?\n1-Yes\n2-No\n");
294. editchoice\_manual:
295. scanf("%d", & EditChoice);
296. if (EditChoice == 1)
297. {
298. EditArray(InitialArray, InitialArraySize);
299. }
300. else if (EditChoice == 2)
301. {
302. }
303. else
304. {
305. printf("Invalid choice!\nEnter 1 or 2\n");
306. goto editchoice\_manual;
307. }
308. PrintArray(InitialArray, InitialArraySize);
309. SecondArraySize = GetMinIndex(InitialArray, InitialArraySize);
310. printf("Min index is %d and InitialArray[%d] = %d\n", SecondArraySize, SecondArraySize, InitialArray[SecondArraySize]);
311. int\* SecondArray = (int\*) malloc(SecondArraySize \* sizeof(int));
312. CopyArrays(InitialArray, SecondArray, SecondArraySize);
313. printf("---------\nNew Array:\n");
314. PrintArray(SecondArray, SecondArraySize);
315. printf("---------\nSorted New Array:\n");
316. quickSort(SecondArray, 0, SecondArraySize - 1);
317. PrintArray(SecondArray,SecondArraySize);
318. printf("----------\n");
319. PrintAllRepNums(SecondArray,SecondArraySize);
320. int save = 0;
321. save\_option:
322. printf("\nWould you like to Save the array?\n1-Yes\n2-No(Exit to Main Menu)\n");
323. scanf("%d", &save);
324. if (save == 1)
325. {
326. SaveArray(SecondArray,SecondArraySize);
327. }
328. else if (save == 2)
329. {
330. exit;
331. }
332. else
333. {
334. printf("Invalid choice!\n");
335. goto save\_option;
336. }
337. }
338. void SaveArray(int array[], int size)
339. {
340. //saves array to an file
341. char name[64];
342. printf("Enter the name of the file to be created: ");
343. scanf("%s", &name);
344. FILE\* fptr;
345. fptr = fopen(name, "w");
346. for (int i = 0; i <= size/4; i++)
347. {
348. for (int j = i\*4; j < i\*4 + 4; j++)
349. {
350. fprintf(fptr,"%d",array[j]);
351. if(j<i\*4 + 3) { fprintf(fptr, " ");}
352. }
353. fprintf(fptr, "\n");
354. }
355. }
356. void EditArray(int\* array, int size)
357. {
358. int ChangeChoiceInFunc = 1;
359. int IndexToEditinFunc = 0;
360. do
361. {
362. EnteringIndexForEditing:
363. printf("What index whould you like to Edit? ");
364. scanf("%d", &IndexToEditinFunc);
365. if (IndexToEditinFunc >=0 && IndexToEditinFunc < size)
366. {
367. printf("InitialArray[%d] = ", IndexToEditinFunc);
368. scanf("%d", &array[IndexToEditinFunc]);
369. editchoice:
370. printf("Change another number?(1/0)\n1-Yes\n0-No\n");
371. scanf("%d", &ChangeChoiceInFunc);
372. if ( !(ChangeChoiceInFunc == 1 || ChangeChoiceInFunc == 0) )
373. {
374. printf("Invalid choice!\n");
375. goto editchoice;
376. }
377. }
378. else
379. {
380. printf("Invalid index!\n");
381. goto EnteringIndexForEditing;
382. }
383. }while(ChangeChoiceInFunc == 1);
384. }