Lab 4: Dynamic Arrays / Mailbox

Thomas Bischoff

Original

Date: 3 / 6 / 2018

Problems 1 – 3:

1. Create a function which will perform *Binary Search* for a value entered in by the user, if the value is found return the index / position that it is located at. Otherwise, if the value cannot be found return a -1.
2. Create a function which will find two values within the array that will combine to equal a user specified sum.
3. Create a function which will go through the array and determine if all the values are unique or if they repeat. If the values are all unique it will return True, otherwise the function will return False.

Pseudocode 1 – 3:

Question 1 = int binary\_search(int \*ptr\_array, int length, int value) AAAAANot Necessary Question 2 = void sum\_finder(int \*ptr\_array, int length, int sum) 1. Create a value named num\_of\_pairs and initialize initialize it to “0”. 2. Generate a For Loop which will start at “0” and increase by “1” each time for as long as the it is less than the length of the array minus “1”. A. Generate a For Loop which will start at the current position of the first For Loop plus “1” and increase by “1” each time for as long it less than the length of the array. + Create a value labeled as addition and set it equal to value contained at location in the array indicated by the first For Loop plus the value contained at the location in the array indicated by the second For Loop. + Check if the value addition is equivalent to the sum provided by the user. - Display the values indicated in the array by both For Loop contained within “()” and separated by “,”. - Add “1” to the value num\_of\_pairs. 3. Check if num\_of\_pairs is equivalent to “0”. A. Display that there are no pairs which equal to given sum.

Pseudocode 1 – 3:

Code 1 – 3:

Question 3 = bool isUnique(int \*ptr\_array, int length) 1. Create value called unique and set it to being true. 2. Generate For Loop which will start at “0” and increase by “1”, repeat this as long as it is less than the length of the array. A. Check if the item in the location indicated by For Loop is equivalent to the next item in the array. (Only Works on a Sorted Array) + Set unique to be equal to false. 3. Return unique.

1. //######################################################################
2. // Program Header: Dynamic\_Array\_Max.cpp
3. // The function of this program is to provide the user with the means to
4. // sort an array using bubble sort, find the highest value in an array,
5. // find an item within an array using binary search, find all pairs of
6. // integers which provide a given sum, and shows if every value in the
7. // array is unique.
8. // Author: Thomas Bischoff
9. // Date Created: 2/27/2018
10. //######################################################################
12. #include <iostream>
13. #include <fstream>
14. **using** **namespace** std;
16. // Function Declarations
17. **int** maxArray(**int** \*ptr\_array, **int** length);
18. **void** bubble\_sort(**int** \*ptr\_array, **int** length);
19. **int** binary\_search(**int** \*ptr\_array, **int** length, **int** value);
20. **void** sum\_finder(**int** \*ptr\_array, **int** length, **int** sum);
21. **bool** isUnique(**int** \*ptr\_array, **int** length);
23. **int** main()
24. {
25. // Ask the User for the Size of the Array
26. **int** size;
27. cout << "Please Enter the Length of the Array: ";
28. cin >> size;
29. // Allocate Dynamic Memory for the Array
30. **int** \*ptr\_ages = **new** **int**[size];
31. // Open Input File for Reading; inp\_file -
32. ifstream inp\_file("input.txt");
33. // Pointer to the First Character in the Input File
34. **for** (**int** i = 0; i < size; i++)
35. {
36. // Read Integers from the File
37. inp\_file >> ptr\_ages[i];
38. }
39. cout << endl;
40. // Call Function to Compute the Max Value
41. **int** maxAge = 0;
42. maxAge = maxArray(ptr\_ages, size);
43. **for** (**int** i = 0; i < size; i++)
44. cout << ptr\_ages[i] << " ";
45. cout << endl;
46. cout << "\nMax Age is " << maxAge << endl;
47. // Call bubble\_sort
48. bubble\_sort(ptr\_ages, size);
49. cout << "Sorted Ages: ";
50. **for** (**int** i = 0; i < size; i++)
51. cout << ptr\_ages[i] << " ";
52. cout << endl;
53. // Call binary\_search
54. **int** users\_value;
55. cout << "Please Enter a Value You Wish to Find: ";
56. cin >> users\_value;
57. **int** location = binary\_search(ptr\_ages, size, users\_value);
58. **if** (location == -1)
59. cout << "Sorry, but " << users\_value << " is Not in the Array." << endl;
60. **else**
61. cout << users\_value << " is Located at " << location << endl;
62. **int** users\_addition;
63. cout << "All Pairs in an Array of Integers Whose Sum is Equal to a Given Value: ";
64. cin >> users\_addition;
65. sum\_finder(ptr\_ages, size, users\_addition);
66. **bool** different = isUnique(ptr\_ages, size);
67. **if** (different == **true**)
68. cout << "The Array is Unique." << endl;
69. **else**
70. cout << "The Array Repeats." << endl;
71. // Have User Create a New Array
72. **int** size\_two;
73. cout << "Enter a Size for an Array: ";
74. cin >> size\_two;
75. // Initialize the Array
76. **int** \*ptr\_users\_array = **new** **int**[size\_two];
77. **for** (**int** i = 0; i < size\_two; i++)
78. {
79. cout << "Enter an Integer for Position " << i << ": ";
80. cin >> ptr\_users\_array[i];
81. }
82. // Call Bubble Sort
83. bubble\_sort(ptr\_users\_array, size\_two);
84. cout << "Sorted Integers: ";
85. **for** (**int** i = 0; i < size\_two; i++)
86. cout << ptr\_users\_array[i] << " ";
87. cout << endl;
88. // Call binary\_search
89. **int** users\_value\_two;
90. cout << "Please Enter a Value You Wish to Find: ";
91. cin >> users\_value\_two;
92. **int** location\_two = binary\_search(ptr\_users\_array, size\_two, users\_value\_two);
93. **if** (location\_two == -1)
94. cout << "Sorry, but " << users\_value\_two << " is Not in the Array." << endl;
95. **else**
96. cout << users\_value\_two << " is Located at " << location\_two << endl;
97. // Sum Finder
98. **int** users\_addition\_two;
99. cout << "All Pairs in an Array of Integers Whose Sum is Equal to a Given Value: ";
100. cin >> users\_addition\_two;
101. sum\_finder(ptr\_users\_array, size\_two, users\_addition\_two);
102. // Is Unique
103. **bool** different\_two = isUnique(ptr\_users\_array, size\_two);
104. **if** (different\_two == **true**)
105. cout << "The Array is Unique." << endl;
106. **else**
107. cout << "The Array Repeats." << endl;
108. }
110. // Function Definition
111. **int** maxArray(**int** \*ptr\_array, **int** length)
112. {
113. // Declare a Variable to Hold the Current Max Value
114. **int** maxSoFar = ptr\_array[0];
115. **for** (**int** i = 1; i < length; i++)
116. {
117. **if** (maxSoFar < ptr\_array[i])
118. maxSoFar = ptr\_array[i];
119. }
120. **return** maxSoFar;
121. }
123. // Function Definition
124. **void** bubble\_sort(**int** \*ptr\_array, **int** length)
125. {
126. **int** nr\_swaps\_per\_pass = 1;
127. **int** index\_end = length - 1;
128. **while** (nr\_swaps\_per\_pass > 0)
129. {
130. nr\_swaps\_per\_pass = 0;
131. **for** (**int** i = 0; i < index\_end; i++)
132. {
133. **if** (ptr\_array[i] > ptr\_array[i + 1])
134. {
135. nr\_swaps\_per\_pass++;
136. // Swap ptr\_array[i] with ptr\_array[i + 1]
137. **int** tmp = ptr\_array[i];
138. ptr\_array[i] = ptr\_array[i + 1];
139. ptr\_array[i + 1] = tmp;
140. }
141. }
142. index\_end --;
143. }
144. }
146. // Function Definition
147. **int** binary\_search(**int** \*ptr\_array, **int** length, **int** value)
148. {
149. **int** minimum = 0;
150. **int** maximum = length - 1;
151. **while** (minimum <= maximum)
152. {
153. **int** median = (minimum + maximum) / 2;
154. **if** (ptr\_array[median] == value)
155. **return** median;
156. **else** **if** (ptr\_array[median] > value)
157. maximum = median - 1;
158. **else**
159. minimum = median + 1;
160. }
161. **int** not\_found = -1;
162. **return** not\_found;
163. }
165. // Function Definition
166. **void** sum\_finder(**int** \*ptr\_array, **int** length, **int** sum)
167. {
168. **int** num\_of\_pairs = 0;
169. **for** (**int** i = 0; i < (length - 1); i++)
170. {
171. **for** (**int** j = i + 1; j < length; j++)
172. {
173. **int** addition = ptr\_array[i] + ptr\_array[j];
174. **if** (addition == sum)
175. {
176. cout << "(" << ptr\_array[i] << ", " << ptr\_array[j] << ") ";
177. num\_of\_pairs += 1;
178. }
179. }
180. }
181. cout << endl;
182. **if** (num\_of\_pairs == 0)
183. cout << "There are No Pairs Which Have the Sum " << sum << endl;
184. }
186. // Function Definition
187. **bool** isUnique(**int** \*ptr\_array, **int** length)
188. {
189. **bool** unique = **true**;
190. **for** (**int** i = 0; i < length; i++)
191. {
192. **if** (ptr\_array[i] == ptr\_array[i + 1])
193. unique = **false**;
194. }
195. **return** unique;
196. }

Sample Output 1 – 3:

Please Enter the Length of the Array: 7 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 18 19 21 17 18 22 19 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa Max Age is 22 Sorted Ages: 17 18 18 19 19 21 22 Please Enter a Value You Wish to Find: 19 19 is Located at 3 All Pairs in an Array of Integers Whose Sum is Equal to a Given Value: 38 (17, 21) (19, 19)

Sample Output 1 – 3:

The Array Repeats. Enter a Size of an Array: 7 Enter an Integer for Position 0: 12 Enter an Integer for Position 1: 14 Enter an Integer for Position 2: -11 Enter an Integer for Position 3: 15 Enter an Integer for Position 4: 17 Enter an Integer for Position 5: 19 Enter an Integer for Position 6: 20 Sorted Integers: -11 12 14 15 17 19 20 Please Enter a Value You Wish to Find: 15 15 is Located at 3 All Pairs in an Array of Integers Whose Sum is Equal to a Given Value: 9 (-11, 20) The Array is Unique.

Problem 4: In a Post Office there are 150 mailboxes, if the second mailbox is opened and then every two after, after that is done start again with the third mailbox and open or close every third mailbox after that, repeat this process until you reach the 150th mailbox and open or close depending on its current state, finally display all the mailboxes which are currently open.

Pseudocode 4:

int main() - 1. Set a constant integer called Number\_of\_Mailboxes to be equal to “150”. 2. Setup an array called mailbox\_array with a length of Number\_of\_Mailboxes. 3. Go through each item in the array to be false, indicating that it is closed.

Pseudocode 4:

int main() - 4. Run opening\_and\_closing\_mailboxes with mailbox\_array and Number\_of\_Mailboxes. 5. Run display\_results with mailbox\_array and Number\_of\_Mailboxes. void opening\_and\_closing\_mailboxes(bool \*ptr\_array, int size) - 1. Set the a value called step to be equal to “2”. 2. Go through each element in the array stating with “1” and increase each time by “1”. A. Go through each element in the array starting at the current position of pervious step, this will increase at a rate of the step. + If the value stored at the given index of the array is false. - If it is change the value of the current position to be true. - Otherwise, convert the value to be false. B. Increase step by “1”. void display\_results(bool \*ptr\_array, int size) - 1. Display a message saying, “Closed Mailboxes:”. 2. Create a value called count and set it equal to “1”. 3. Go through each element in the array starting with “0” and increase by “1” each time. A. Check if the value at the current index is false. + Display the current count with the word “Mailbox” and the current index number. + Add “1” to the value count.

Code 4:

1. //######################################################################
2. // Program Header: mailbox\_problem.cpp
3. // The function of this program is to state which mailbox are closed at
4. // the end of the day.
5. // Author: Thomas Bischoff
6. // Date Created: 2/27/2018
7. //######################################################################
9. #include <iostream>
10. **using** **namespace** std;
12. // Function Declarations:
13. **void** opening\_and\_closing\_mailboxes(**bool** \*ptr\_array, **int** size);
14. **void** display\_results(**bool** \*ptr\_array, **int** size);
16. **int** main()
17. {
18. // Provide the Number of Mailboxes Avalible
19. **const** **int** Number\_of\_Mailboxes = 150;
20. // Initialize an Array
21. **bool** \*mailbox\_array = **new** **bool**[Number\_of\_Mailboxes];
22. // Initialize the Elements Within the Array
23. **for** (**int** element = 0; element < Number\_of\_Mailboxes; element++)
24. {
25. // When an Element in mailbox\_array is False this Means it is Closed
26. mailbox\_array[element] = **false**;
27. }
28. // Run opening\_and\_closing\_mailboxes Function
29. opening\_and\_closing\_mailboxes(mailbox\_array, Number\_of\_Mailboxes);
30. // Run display\_results Function
31. display\_results(mailbox\_array, Number\_of\_Mailboxes);
32. }
34. **void** opening\_and\_closing\_mailboxes(**bool** \*ptr\_array, **int** size)
35. {
36. // Intialize the Step Value
37. **int** step = 2;
38. // Go Through Each "Mailbox" Starting with One
39. **for** (**int** element\_1 = 1; element\_1 < size; element\_1++)
40. {
41. // Increment the Distance Traveled in Each Pass by the Starting Mailboxes Number
42. **for** (**int** element\_2 = element\_1; element\_2 < size; element\_2 += step)
43. {
44. // If the Mailbox is Closed, Open it
45. **if** (ptr\_array[element\_2] == **false**)
46. ptr\_array[element\_2] = **true**;
47. // If the Mailbox is Open, Close it
48. **else**
49. ptr\_array[element\_2] = **false**;
50. }
51. // Add One to Step
52. step += 1;
53. }
54. }
56. **void** display\_results(**bool** \*ptr\_array, **int** size)
57. {
58. // Display Introduction to the List of Closed Mailboxes
59. cout << "Closed Mailboxes:" << endl;
60. // Initialize a Count Value for Numbering of the List
61. **int** count = 1;
62. // Go Through the Array Starting with Element 0
63. **for** (**int** element = 0; element < size; element++)
64. {
65. // If the Mailbox is Closed Display its Index Number
66. **if** (ptr\_array[element] == **false**)
67. {
68. // Display the Mailboxes Index
69. cout << count << ". " << "Mailbox " << element << endl;
70. // Add One to the Current Count
71. count += 1;
72. }
73. }
74. }

Sample Output 4:

Closed Mailboxes: 1. Mailbox 0 2. Mailbox 3 3. Mailbox 8 4. Mailbox 15 5. Mailbox 24 6. Mailbox 35 7. Mailbox 48 8. Mailbox 63 9. Mailbox 80 10. Mailbox 99 11. Mailbox 120 12. Mailbox 143