**Data Structures and Algorithms**

**Lab 4**

**Submitted To:**

Mr. Dilshad Sabir

**Submitted By:**

Manaal Waseem

FA18-BCE-074

**In Lab:**

**Task 1:**

***Debugging code for errors*.**

**• Make a new project and add these files to the project.**

**• Compile and run the program**

**• Choose to load the list from the file**

**Now you will observe that some of the functions do not work as intended. Figure out the problems (by debugging the code) and correct them so that you have a correctly running implementation of Circular Doubly Linked List. Report the errors that you have corrected.**

**Program:** In this program, already provided code has been compiled and its working has been analyzed. The function ***“listLength”*** has been implemented by calling it in *‘main.c’* and passing first node **‘head’** as argument to the function. As a result; length of list is printed on the console.

**Task 2:**

**Implementing Node Removal and Node Insertion Tasks**

**Use the created circular linked list in Task 01 and add functions for**

**1*. void deleteNodeFromStart(struct node\_d \*\* head)***

**2. *int deleteNodeAfter(struct node\_d \* head, int idx)***

**3. *void insertNodeAtStart(struct node\_d \*\* head)***

**Moreover write a function *‘void printMemMap(struct node\_d \* head)*’ that prints out the memory map of the linked list.**

**Program:** In this program, a function ***“searchNode”*** takes first node of the linked list and the reference age as its argument. Variables **‘i’** and **‘flag’** are initialized as **1**and **false** respectively. A pointer *‘current’* of type **struct employee** is initialized with the first node of list. Function ***“isEmpty”*** checks if the list is empty otherwise while loop iterates till **‘next’** pointer of *‘current’* doesn’t point to **NULL** and checks if **‘age’** of *‘current’* is equal to *‘key’* i.e. reference age entered by user; **flag** is marked **‘true’** and position of respective node is printed on the console depending upon variable **‘i’**. If **flag** remains 0, a message of required node not found is displayed on the console.

**‘void deleteNodeFromStart(struct node\_d \*\* head)’**

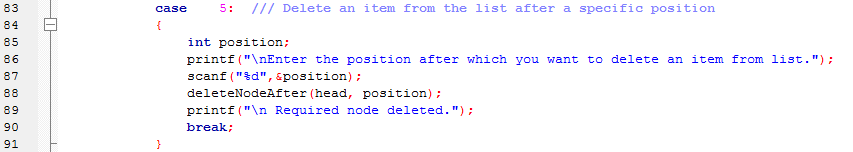


main.c



DoublyLinkedList.c

**‘int deleteNodeAfter(struct node\_d \* head, int idx)’**



main.c



DoublyLinkedList.c

**‘void insertNodeAtStart(struct node\_d \*\* head)’**

****

main.c



DoublyLinkedList.c

**‘void printMemMap(struct node\_d \* head)’**



main.c

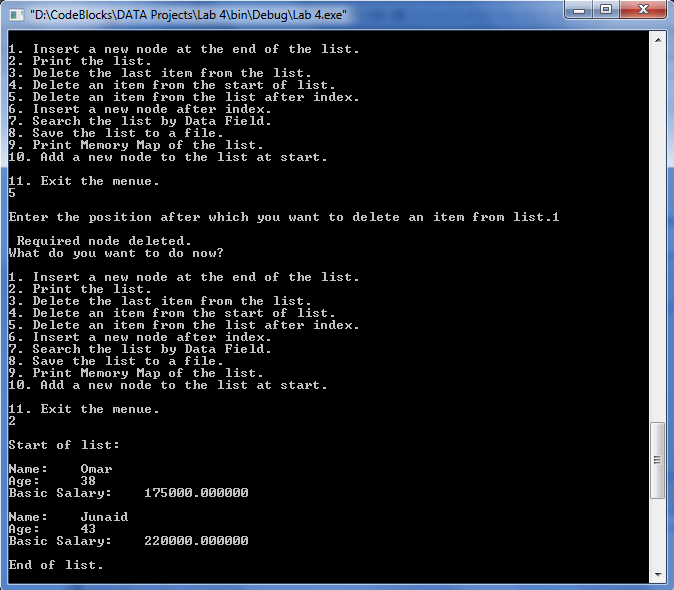


DoublyLinkedList.c

**Output:**

****

**‘Delete from beginning’**



**‘Delete after’**

****

**Post Lab:**

**Learn more about at the Josephus Problem from the following links and make a**

**Circular Doubly Linked List simulation of this problem. (Your program should print the remaining people in each iteration).**

**Program:** In this program, the user is prompted to enter the total number of soldiers and the number of soldiers to be skipped. Function ***‘CreateList’*** creates as many nodes as the total number of soldiers. The data for each node is printed on console. Then function, ***‘getJosephusPosition’***

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <stdbool.h>

4 #include <string.h>

5

6 **struct** node\_d

7 {

8

9 **int** data;

10

11 **struct** node\_d \* next;

12 **struct** node\_d \* prev;

13 };

14

15 **bool** isListEmpty(**struct** node\_d \* head);

16 **void** CreateList(**struct** node\_d \*\* head, **int** n);

17 **void** printList(**struct** node\_d \* head);

18 **int** getListLength(**struct** node\_d \* head);

19 **void** getJosephusPosition(**struct** node\_d \* head, **int** m, **int** n);

20 **void** flush();

21

22 **int** main()

23 {

24 **struct** node\_d \* head = NULL; **/// Pointer to the first node (head) of the list**

25

26 **int** m,n;

27 **int** list\_length;

28

29 printf("\n Enter the total number of soldiers \n");

30 scanf("%d", &n);

31 printf("\n Enter the number of soldiers to be skipped... \n");

32 scanf("%d", &m);

33 flush();

34

35 CreateList(&head,n);

36 printList(head);

37

38 getJosephusPosition(head, m, n);

39

40 }

41 **/\*\* Tests if the list is empty. ONLY the head of the list**

42 **should be passed to this function.**

43 **\*/**

44 **bool** isListEmpty(**struct** node\_d \* head)

45 {

46 **return** (head == NULL);

47 }

48

49 **/\*\* Insert a new node at the last location. i.e. after the current one.**

50 **If the list is currently empty, a new node is created for the head.**

51 **Note: This function takes an argument of type pointer to pointer.**

52 **\*/**

53 **void** CreateList(**struct** node\_d \*\* head, **int** n)

54 {

55

56 **struct** node\_d \* temp; // = \*head;

57

58 **for**(**int** i=1; i<=n; i++)

59 {

60 temp = \*head;

61 **///create a new node**

62 **struct** node\_d \* new\_node = (**struct** node\_d\*) malloc(**sizeof**(**struct** node\_d));

63 **/// get data for the newly created node.**

64 new\_node->data=i;

65 **if**(isListEmpty(\*head)) **/// if currently the list is empty**

66 {

67 \*head = new\_node;

68 **///point its next and previous pointers to the start**

69 new\_node->next = \* head;

70 new\_node->prev = \* head;

71 }

72 **else**

73 {

74 **while**((temp->next) != \*head)

75 temp = temp->next; **/// scroll to the end of the list**

76 temp->next = new\_node;

77 new\_node->next = \* head; **/// Point to the first location**

78 new\_node->prev = temp; **/// Point to the second last item**

79 }

80 }

81

82

83 }

84

85 **/\*\* Function to display the data of all the records**

86 **in the list.**

87 **\*/**

88 **void** printList(**struct** node\_d \* head)

89 {

90 **struct** node\_d \* ptr = head;

91 printf("\nStart of list: \n");

92

93 **if**(isListEmpty(head))

94 {

95 printf("\nList is empty: \n");

96 **return**;

97 }

98 **///start from the beginning and go till the last node**

99 **do**

100 {

101 printf("\n The data stored is %d.\n", ptr->data);

102 ptr = ptr->next;

103 }

104 **while**(ptr != head);

105

106 printf("\nEnd of list.\n");

107 }

108

109 **/\*\* Function to find the length (in number of nodes) of the list.**

110 **\*/**

111 **int** getListLength(**struct** node\_d \* head)

112 {

113 **int** length = 0;

114 **struct** node\_d \* current;

115

116 **if**(isListEmpty(head))

117 **return**(0);

118

119 **for**(current = head; current->next != head; current = current->next)

120 {

121 length++;

122 }

123

124 **return** (length+1);

125 }

126

127 **/\*\* Function to find the only person left**

128 **after one in every m-th node is killed**

129 **in a circle of n nodes \*/**

130 **void** getJosephusPosition(**struct** node\_d \* head, **int** m, **int** n)

131 {

132 **struct** node\_d \*temp= head;

133

134 **/\*\* while only one node is left in the**

135 **linked list\*/**

136 **struct** node\_d \*ptr2 = head;

137 **struct** node\_d \*toDelete = head;

138 **int** no\_iteration = 0;

139

140 **while** (temp->next != temp)

141 {

142 ++no\_iteration;

143 printf("\nNumber of remaining people in iteration %d is %d .\n", no\_iteration, getListLength(temp));

144 **/// Find m-th node**

145 **int** count = 1;

146 **while** (count != m)

147 {

148 ptr2 = temp;

149 temp = temp->next;

150 count++;

151 }

152

153 **/\*\* Remove the m-th node \*/**

154 ptr2->next = temp->next;

155 toDelete = temp;

156 temp = ptr2->next;

157 temp->prev=ptr2;

158 free(toDelete);

159 }

160

161 printf ("Last person left standing "

162 "(Josephus Position) is %d\n ",

163 temp->data);

164 }

165

166 **void** flush()

167 {

168 **int** c;

169 **while** ((c = getchar()) != '\n' && c != EOF);

170 }

**Output:**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**THE END**