Programming Assignment 4

Linked Lists

(Deadline: Nov 7, 23:59)

Instructions for the output: The below instructions apply to all the programs (even for the upcoming labs).

While printing the output, there is no new line

1. between any two output lines
2. after the last output line
3. Before the first input line

Refer to Appendix 1 for more details.

If the program fails the sanity check, it should just print **"error",** nothing less or more. Please note that "error" is different from "ERROR" and "Error" in C.

==

==

REGULAR EXERCISES

There are 3 regular exercises. You need to make only one program for all the 3 exercises. Since the exercises differ in the number of input parameters, the first argument to the program should be the exercise number. Refer to the test cases for further clarification.

Sanity checks required:

1. The number of input parameters should be in accordance with the exercise number.
2. You can assume the input parameters to be integers. However, you should verify them for their sign (positive/negative).

==

Exercise 1. Write a program to determine whether the number of nodes in a linked list is even or odd.

Input format: ./a.out [exercise number] [Elements of the linked list]

Output format: 0 if the number of elements is even, 1 if the number of elements is odd.

Test Case1: ./a.out 1 1 2 3 4 5 *(First argument is the exercise number)*

1

Test Case2: ./a.out 1 3 18 200 200 80 0

0

Test Case 3: ./a.out 1 1

1

Test Case 4: ./a.out 1

error *(no elements in the linked list. You can also print a "#' or 0 here as an output)*

Test Case 5: ./a.out

error *(incorrect number of arguments)*

===

Exercise 2. Write a program to print the prime numbers present in the given linked list.

Input format: ./a.out [exercise number] [Elements on the linked list]

Output format: Prime numbers in the given linked list, separated by space

Test Case1: ./a.out 2 2 6 9 13 15 *(First argument is the exercise number)*

2 13

Test Case2: ./a.out 2 2 4 6 10 16

2

Test Case 3: ./a.out 2 4 6 8 23 13 30

23 13

Test Case 4: ./a.out 2 4 6 8 22 14 30 *(No output)*

#

Test Case 5: ./a.out 2 *(Empty List : No output)*

#

===

Exercise 3. Write a program to calculate the sum of the elements present in the given linked list.

Input format: ./a.out [exercise number] [elements of the linked list]

Output format: Sum of the elements

Test Case1:

./a.out 3 9 2 64 7 83 67 7 3 9 -15

236

Test Case2: ./a.out 3 7 5 7 346 1 2 19 17

404

Test Case 3: ./a.out 3 -10 10 *(sum is zero)*

0

Test Case 4: ./a.out 3*(Empty linked list, Even a "#" will be acceptable here as the output)*

0

==
==

BONUS EXERCISES

There are 2 bonus exercises. Create separate programs for them.

Sanity checks required:

1. You can assume the input parameters to be integers. However, you should verify them for their sign (positive/negative).

Exercise 1: Given as input, a linked list and several pairs of elements. Process these pairs in such a way that the node with the first element of the pair starts pointing to the node with the second element of the pair. Repeat this process for each pair. Print the resultant linked list.

1. The pairs should be processed sequentially in the given order.
2. If while processing a pair, the linked list breaks, you can discard the disconnected nodes. Thereafter, if a pair contains an element from the disconnected nodes, print "error'.
3. If there are duplicate entries in the linked list, consider the first occurrence (while traversing the linked list <u>(in its current form)</u> starting from head till the end).
4. It is so possible that **after processing all the pairs**, the linked list develops cycles . In that case, print "error". It is so possible that after processing a pair, the linked list becomes invalid, however while processing a later pair, it again becomes valid. Hence, check the validity of the linked list only after processing all the pairs.
5. A '#' as the first element in a pair represents that the node with this element should be made the head "# 5" means that the node with the element 5 is now the head of the list.
6. A '#' as the second element in a pair represents null, i.e., the entry "5 #" means that the node with the element 5 will now have "null" in its "next" field.

Input format: ./a.out [number of elements in the linked list] [Elements of the linked list separated with space] [Pair 1] [Pair 2]... [Pair k]

Output format: Elements of the modified linked list separated by space

Test Case 1: ./a.out 7 6 78 67 12 56 67 80 12 67 67 78 67 6 # 67 78 #

67 6 78

Following is the explanation for the changes made to the linked list after every operation.

6 -> 78 -> 67 -> 12 -> 56 -> 67 -> 80

After processing (12,67) : 6 -> 78 -> 67 <-> 12 (A cycle develops between 12 and 67)

After processing (67,78) : 6 -> 78 <-> 67 (12 becomes disconnected from the linked list)

*After processing (67,6) : **6** -> 78 -> 67 -> **6** ->...   (back link from 67 to 6. Same node is represented by the same color)*

*After processing (#, 67) : **67** -> 6 -> 78 -> **67** ->...   (67 becomes head)*

After processing (78, #) : 67 -> 6 -> 78   (78 becomes the last node)

Test Case 2: ./a.out 4 12 20 21 40 20 12 21 #   *(Processing the first pair, 20 12 breaks 21 and 40 from the linked list, hence error while processing the second pair, 21 #)*

error

Test Case 3: ./a.out 2 -3 4 1 2   *(Elements given in the pair do not exist in the linked list)*

error

Test Case 4: ./a.out 6 1 2 3 4 5 6 6 5 5 4 4 3 3 2 2 1 # 6   (*Cycle)*

error

Test Case 5: ./a.out 6 1 2 3 4 5 6 6 5 5 4 4 3 3 2 2 1 # 6 1 #

error

Test Case 6: ./a.out 7 6 78 67 12 56 67 80 12 67 6 67 67 # # 67

67

Test Case 7: ./a.out 7 6 78 67 12 56 67 80 80 67 67 56 67 78 78 6

error   *(Cycle)*

==

Exercise 2: Consider any phonebook application where the goal is to store and retrieve phone numbers efficiently. This can be done using linked lists. The design of the phonebook will be discussed in the TA session.

Input format: ./a.out [number of LSBs] [Mobile numbers to be inserted separated by space] [Array index, i, corresponding to which the linked list is to be printed]

Output format:

[Linked list corresponding to A[i]]   (Print “#” is the linked list is empty)

Test Case 1: ./a.out 1 9627316384 9361824537 8362176345 7546329102 6351829374 8325173526 9563187253 8623156283 3

9563187253 8623156283

Test Case 2: ./a.out 2 9627316384 9361824537 8362176345 7546329102 6351829374 8325173526 9563187253 8623156283 45

8362176345

Test Case 3: ./a.out 2 9627316384 9361824537 8362176345 7546329102 6351829374 8325173526 9563187253 8623156283 73 *(The linked list corresponding to A[73] is empty)*

#

Test Case 4: ./a.out 2 9627316384 9361824537 8362176345 7546329102 6351829374 8325173526 9563187253 8623156283 873 *(There are only 100 linked lists)*

error

==
==

<u>SUBMISSION INSTRUCTIONS</u>

Create separate folders for the submission of regular problems and the bonus problems. Name the folders as [your_roll_number]_[your_name]_regular and [your_roll_number]_[your_name]_bonus, respectively.

[your_roll_number]_[your_name]_regular: This folder should have one file named [your_roll_number]_[your_name]_p1.c.

[your_roll_number]_[your_name]_bonus: This folder should have two files named [your_roll_number]_[your_name]_p2.c (exercise 1) and [your_roll_number]_[your_name]_p3.c (exercise 2).

Zip both the folders together and name the zip file as [your_roll_number]_[your_name]_lab4. Upload the zip file to the Google classroom. Kindly use your complete roll number as assigned by the institute.

==

==

Appendix 1

Your program should not print any unnecessary new lines (or even tabs and spaces). As an example, assume that your code is printing 2 numbers as output, n (having value 10) and m (having value 40). Then your sequence of printf statements should be

==

```
printf("%d",n);
printf("\n");
printf("%d",m);
```

This will result in an output like

```
10
40
```

==

and not any of the following

1.

```
printf("%d",n);
printf("\n\n");
printf("%d",m);
```

Here, there is an additional newline between two output lines.

This will result in an output like

```
10

40
```

==

2.

```
printf("%d",n);
printf("\n");
printf("%d",m);
printf("\n");
```

Here, there is an additional newline after the last output line.

This will result in an output like

```
10
40

```

==

3.

```
printf("\n");
printf("%d",n);
printf("\n");
printf("%d",m);
```

Here, there is an additional newline before the first output line.
This will result in an output like

```

10
40
```

=======================================