National Institute of Technology Calicut
Department of Computer Science and Engineering
Third Semester B. Tech.(CSE)
CS2092D Programming Laboratory
Assignment #5

Submission deadline (on or before):

- 10.10.2022, 11:55 PM

Policies for Submission and Evaluation:

- You must submit your assignment in the Eduserver course page, on or before the submission deadline.
- Ensure that your programs will compile and execute without errors using gcc compiler.
- During the evaluation, failure to execute programs without compilation errors may lead to zero marks for that evaluation.
- Detection of ANY malpractice related to the lab course can lead to awarding an F grade in the course.

Naming Conventions for Submission

- The source codes must be named as

  `ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME>_<PROGRAM-NUMBER>.c`

  (For example: $ASSG1\_BxxyyyyCS\_LAXMAN\_1.c$).

  Create separate files for each question and place all of them in a single folder named

  `ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME>`

  (Example: $ASSG1\_BxxyyyyCS\_LAXMAN$). Then Zip the **folder**.

- Submit the single ZIP (.zip) file you have created(do not submit in any other archived formats like .rar, .tar, .gz). The name of this file must be

  `ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME>.zip`

  (Example: $ASSG1\_BxxyyyyCS\_LAXMAN.zip$). DO NOT add any other files (like temporary files, input files, etc.) except your source code, into the zip archive.

  If you do not conform to the above naming conventions, your submission might not be recognized by our automated tools, and hence will lead to a score of 0 marks for the submission. So, make sure that you follow the naming conventions.

Standard of Conduct

- Violation of academic integrity will be severely penalized. Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work MUST BE an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or evaluation and will be reported to the department council for record keeping and for permission to assign F grade in the course. The department policy on academic integrity can be found at: https://minerva.nitc.ac.in/?q=node/650.

General Instructions

- Programs should be written in C language and compiled using gcc compiler. **Submit the solutions to the questions through the submission link in Eduserver**.
- Check your programs with sufficiently large values of inputs with in the range as specified in the question.
- Global and/or static variables should not be used in your program.
- You shouldn't change the function prototype specified in the question.

QUESTIONS

1. A Singly linked list L is a data structure in which the objects are arranged in a linear order. Each node of a Singly linked list L is an object with a data attribute, *key* and a pointer attribute, *next*. Given a node p in the list, $p.next$ points to its successor in the linked list. An attribute $L.head$ points to the first node of the list.
   Write a menu driven program to implement an unsorted Singly linked list L. Each node of L contains a character variable, *key*, to store the data and a pointer variable, *next*, to store the address of the succeeding node in the list. Your program must contain the following functions:

   (In the function prototypes, L and k denote a Singly Linked List and a character belonging to $[A-Z, a-z, 0-9]$ respectively. All operations should be done in a single pass.)

   - MAIN() - repeatedly reads a character 'f', 't', 'i', 'l', 'p' or 'e' from the terminal and calls the sub-functions appropriately, as given in Input format, until character 'e' is encountered.
   - CREATE_NODE(k) - creates a new node with k as the *key*, and *next* points to `NULL`. This procedure returns a pointer to the new node.
   - LIST_INSERT_FRONT(L, k) - inserts a node with key k as the first node of L.
   - LIST_INSERT_TAIL(L, k) - inserts a node with key k as the last node of L.
   - LIST_DELETE_FIRST(L) - deletes the first node from L and prints the deleted node's *key*.
   - LIST_DELETE_LAST(L) - deletes the last node from L and prints the deleted node's *key*.
   - PRINT_LIST(L) - prints the key values of all the nodes in L in order, starting from $L.head$.

   Note:- For every INSERT operation, the new node should be created by calling CREATE_NODE() function.

   Input format:

   - Each line starts with a character from 'f', 't', 'i', 'l', 'p' or 'e' followed by zero, one or two additional characters. The additional characters, if given, are in the range $[A-Z, a-z, 0-9]$.
   - Character 'f' is followed by another character, k separated by a space. For this option, the node with k as the *key* should be inserted to the front of L by calling the function LIST_INSERT_FRONT(L, k).
   - Character 't' is followed by another character, k separated by a space. For this option, the node with k as the *key* is inserted to the tail of L by calling the function LIST_INSERT_TAIL(L, k).
   - Character 'i' is to delete the first node from L and to print the deleted node's key by calling the function LIST_DELETE_FIRST(L).
   - Character 'l' is to delete the last node from L and and to print the deleted node's key by calling the function LIST_DELETE_LAST(L).
   - Character *'p'* is to print all the keys in L by calling the function PRINT_LIST(L).
   - Character 'e' is to 'exit' from the program.

   Output Format:

   - The output (if any) of each command should be printed on a separate line.
   - For options 'i', 'l' and 'p', print *NULL*, if L is empty.

Sample Input:

```
t x
t 9
p
l
i
p
f B
i
l
e
```

Sample Output:

```
x 9
9
x
NULL
B
NULL
```

2. Add the following functionalities to the singly linked list L you created in Question 1. You may modify the MAIN() function as described below to include the additional features. (In the function prototypes, L denotes a Singly Linked List and and k and n denote characters belonging to $[A-Z, a-z, 0-9]$. All operations should be done in a single pass.)

   - MAIN() - repeatedly reads a character 'f', 't', 'a', 'b', 'd', 'i', 'l', 'p' or 'e' from the terminal and calls the sub-functions appropriately, as given in Input format, until character 'e' is encountered.
   - LIST_SEARCH(L, k) - searches for the first occurrence of the key k in L by doing a simple linear search and, if found, returns the address of that node. If a node with key k is not present in the list, or if the list is empty, then the procedure returns the *NULL* pointer.
   - LIST_INSERT_AFTER(L, k, n) - inserts a node with key k just after the first occurrence of the key n in L.
   - LIST_INSERT_BEFORE(L, k, n) - inserts a node with key k just before the first occurrence of the key n in L.
   - LIST_DELETE(L, k) - deletes the node with key k from L and prints the key of the node next to the deleted node. Prints *LAST*, if the deleted node was the last node of L.

Note: - For every INSERT operation, the new node should be created by calling CREATE_NODE() function.

- In LIST_DELETE(L, k), LIST_INSERT_AFTER() and LIST_INSERT_BEFORE() functions, you may call the LIST_SEARCH() function to get the address of the node corresponding to the value n.

Input format:

- Each line starts with a character from 'f', 't', 'a', 'b', 'd', 'i', 'l', 'p' or 'e' followed by zero, one or two additional characters. The additional characters, if given, are in the range $[A-Z, a-z, 0-9]$. For the lines starting with letters 'f', 't', 'i', 'l', 'p' and 'e', the input format is same as those described in Question 1.
- Character 'a' is followed by two more characters, k and n, separated by a space. For this option, the node with k as the key is inserted just after the node with n as the key by calling the function LIST_INSERT_AFTER(L, k, n).
- Character 'b' is followed by two more characters, k and n, separated by a space. For this option, the node with k as the key is inserted just before the node with n as the key by calling the function LIST_INSERT_BEFORE(L, k, n).
- Character 'd' is followed by a character, k, separated by a space. For this option, the node with k as the key is deleted from L and the key of the node next to the deleted node is printed. You may call the function LIST_DELETE(L, k) for this option.

Output format:

- The output (if any) of each command should be printed on a separate line.
- For options 'f', 't', 'i', 'l', 'p' and 'e', do the operations as described in Question 1.
- For options 'i', 'l' , 'p', 'a', 'b' and 'd' print *NULL*, if L is empty.
- For options 'a', 'b' and 'd' print *ABSENT*, if a node with the input key is not present in L.

Sample Input:

```
f x
t 9
a a 9
b B a
p
d x
i
l
a T 9
d B
b Q c
e
```

Sample Output:

```
x 9 B a
9
9
a
ABSENT
LAST
NULL
```

3. A Doubly linked list L is a data structure in which the objects are arranged in a linear order. Each node of L is an object with a data attribute *key* and two pointer attributes- *next* and *prev*. Given a node p in the list, *p.next* points to its successor in the linked list, and *p.prev* points to its predecessor. An attribute *L.head* points to the first node of the list.

   Write a menu driven program to implement an unsorted DOUBLY LINKED LIST L. Your program must contain the following functions: (In function prototypes, L denotes a Doubly Linked list and k, n etc are integers representing the key values of the nodes in L respectively. All operations should be done in a single pass and all keys are distinct).

   - MAIN() - repeatedly reads a character 'f', 't', 'a', 'b', 'd', 'i', 'l', 'r' or 'e' from terminal and calls the sub-functions appropriately, as given in Input format, until character 'e' is encountered.
   - CREATE_NODE(k) - creates a new node with k as the *key*, and *next* and *prev* points to `NULL`. This procedure returns a pointer to the new node.
   - LIST_SEARCH(L, k) - searches for the key k in L by doing a simple linear search and, if found, returns the address of that node. If a node with key k is not present in the list, or if the list is empty, then the procedure returns the *NULL* pointer.
   - LIST_INSERT_FRONT(L, k) - inserts a node with key k as the first node of L.
   - LIST_INSERT_TAIL(L, k) - inserts a node with key k as the last node of L.
   - LIST_INSERT_AFTER(L, k, n) - inserts a node with key k just after the node with key n in L.
   - LIST_INSERT_BEFORE(L, k, n) - inserts a node with key k just before the node with key n in L.
   - LIST_DELETE(L, k) - deletes the node with key k from L and prints the key of the node previous to the deleted node. Prints *FIRST*, if the deleted node was the first node of L.
   - LIST_DELETE_FIRST(L) - deletes the first node from L and prints the deleted node's *key*.
   - LIST_DELETE_LAST(L) - deletes the last node from L and prints the deleted node's *key*.
   - PRINT_REVERSE(L, n) - finds the node x with key n and prints all the keys of nodes from x to the head of L, in that order.

Note: - For every INSERT operation, the new node should be created by calling CREATE_NODE() function.

- In PRINT_REVERSE(), LIST_INSERT_AFTER() and LIST_INSERT_BEFORE() functions, you may call the LIST_SEARCH() function to get the address of the node corresponding to the value n.

Input format:

- Each line contains a character from 'f', 't', 'a', 'b', 'd', 'i', 'l', 'r' or 'e' followed by zero, one or two integers. The integers, if given, are in the range $[-10^6, 10^6]$.
- Character 'f' is followed by an integer, k separated by a space. For this option, the node with k as the *key* should be inserted at the front of L by calling the function LIST_INSERT_FRONT(L, k).
- Character 't' is followed by an integer, k separated by a space. For this option, the node with k as the *key* is inserted to the tail of L by calling the function LIST_INSERT_TAIL(L, k).
- Character 'a' is followed by two integers, k and n, separated by a space. For this option, the node with k as the key is inserted just after the node with n as the key by calling the function LIST_INSERT_AFTER(L, k, n).
- Character 'b' is followed by two more characters, k and n, separated by a space. For this option, the node with k as the key is inserted just before the node with n as the key by calling the function LIST_INSERT_BEFORE(L, k, n).

- Character 'd' is followed by an integer, k, separated by a space. For this option, the node with k as the key is deleted from L and the key of the node previous to the deleted node is printed. You may call the function LIST_DELETE(L, k) for this option.
- Character 'i' is to delete the first node from L and print the deleted node's key by calling the function LIST_DELETE_FIRST(L).
- Character 'l' is to delete the last node from L and print the deleted node's key by calling the function LIST_DELETE_LAST(L).
- Character 'r' is followed by an integer, n, separated by a space. This operation finds the node x with n as the key and prints all the keys from x to $L.head$ in order by calling the function PRINT_REVERSE(L, n).
- Character *'e'* is to 'exit' from the program.

Output Format:

- The output (if any) of each command should be printed on a separate line.
- For options 'i', 'l', 'r', 'a', 'b' and 'd' print *NULL*, if L is empty.
- For options 'a', 'b', 'r' and 'd' print *ABSENT*, if a node with the input key is not present in L.

Sample Input:

```
f 20
t 38
a 22 20
b 35 22
r 38
d 38
i
i
d 33
r 40
d 22
b 20 0
e
```

Sample Output:

```
38 22 35 20
22
20
35
ABSENT
ABSENT
FIRST
NULL
```

4. Given a doubly linked list L with n nodes, each node of L contains an integer value as the *key*. Write a menu driven program to print all the keys in L, starting from the tail end after swapping the key values of x^{th} node from the beginning and the y^{th} node from the end.

   Note: Reuse the definitions and required functions from Question 3 to solve this problem.

   Input format:

   - Each line contains a character from 'f', 't', 'p' or 'e' followed by zero, one or two integers. The integers, if given, are in the range $[-10^6, 10^6]$. For the lines starting with letters 'f', 't' and 'e', the input format is same as those described in Question 3.
   - Character 'p' is followed by two integers, x and y, separated by a space. For this option, print all the keys in L in order, starting from the tail end after swapping the key values of x^{th} node from the beginning and the y^{th} node from the end.

   Output format:

   - The output (if any) of each command should be printed on a separate line.
   - For option p print *NULL*, if L is empty or if x or y is greater than the total number of nodes in L.

Sample Test Cases:

Input #1:

```
f 10
t 11
t 15
t 26
t 24
t 35
t 47
p 5 4
f 0
p 2 2
e
```

Output #1:

```
47 35 26 24 15 11 10
44 10 26 24 15 11 35 0
```

Input #2:

```
p 1 10
f -515
t -211
t 718
f 914
t 413
f -518
p 4 3
e
```

Output #2:

```
NULL
413 718 -211 -515 914 -518
```