National Institute of Technology Calicut
Department of Computer Science and Engineering
Third Semester B.Tech.(CSE)
CS2092D Programming Laboratory
Assignment #5: Part-1

Submission deadline (on or before):

- 27.10.2021, 9:00 AM

Policies for Submission and Evaluation:

- You must submit your assignment (**both parts, Part-1 and Part-2 together**) 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

- Submit a single ZIP (.zip) file (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: *ASSG5_BxxyyyyCS_LAXMAN.zip*). DO NOT add any other files (like temporary files, input files, etc.) except your source code, into the zip archive.

- The source codes must be named as

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

  (For example: *ASSG5_BxxyyyyCS_LAXMAN_1.c*). 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 here.

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.

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 an attribute *key* and one pointer attribute, *next*. Given a node x in the list, $x.next$ 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. x and y represent nodes of L. 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 until character 'e' is entered.
   - CREATE_NODE(k) - creates a new node with *key* k, and a pointer *next*, points to NULL. This procedure returns a pointer to the new node.
   - LIST_SEARCH(L, k) - searches for a node with key k in L by doing a simple linear search, and if found, returns the pointer to the first occurrence of the 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, x) - inserts x to the front of L.
   - LIST_INSERT_TAIL(L, x) - inserts x as the last node of L.
   - LIST_INSERT_AFTER(L, x, y) - inserts the node x after the node y in L.
     Hint: First invoke LIST_SEARCH() function to locate the node y.
   - LIST_INSERT_BEFORE(L, x, y) - inserts the node x before the node y in L.
     Hint: First invoke LIST_SEARCH() function to locate the node y. Insertion of a node x before a node y can be done by locally storing a pointer to the current node before moving to the next node.
   - LIST_DELETE(L, x) - deletes node x from L.
     Hint: First invoke LIST_SEARCH() function to locate the node x.
   - LIST_DELETE_FIRST(L) - deletes the first node from L and print the deleted node's key.
   - LIST_DELETE_LAST(L) - deletes the last node from L and print the deleted node's key..
   - PRINT_LIST(L) - prints the key values of all the nodes in L, starting from the head.

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

   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 $[A-Z, a-z, 0-9]$.
   - Character 'f' is followed by another character separated by a space. In this operation, the node with this character as key is inserted to the front of L by calling the function LIST_INSERT_FRONT(L, x).
   - Character 't' is followed by another character separated by a space. In this operation, the node with this character as key is inserted to the tail of L by calling the function LIST_INSERT_TAIL(L, x).
   - Character 'a' is followed by two more characters separated by a space. In this operation, the node with the first character as key is inserted after the node with second character as key by calling the function LIST_INSERT_AFTER(L, x, y).

- Character 'b' is followed by two more characters separated by a space. In this operation, the node with the first character as key is inserted before the node with second character as key by calling the function LIST_INSERT_BEFORE(L, x, y).
- Character 'd' is followed by a character separated by a space. In this operation, the node with this character as key is deleted from L and the key of the node next to the deleted node is printed. Print -2, if the deleted node was the last node of L. You can call the function LIST_DELETE(L, x) 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 *'p'* is to print all the keys in L, starting from head. If L is empty, print "NULL", without the quotes 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 'd', 'i' , 'l' , if a node with the input key is not present in L or if L is empty, print -1.

Sample Input :

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

Sample Output:

```
x 9 B a
9
9
a
-1
-2
```

2. 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 an attribute *key* and two other pointer attributes: *next* and *prev*. Given a node x in the list, $x.next$ points to its successor in the linked list, and $x.prev$ 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, k, x and y denote a Doubly Linked list, an integer and the nodes of 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 until character 'e' is entered.

- CREATE_NODE(k)- Creates a new node with *key* k and the pointers *next* and *prev* points to NULL. This procedure returns a pointer to the new node.
- LIST_SEARCH(L, k) - searches for a node with key k in L by a simple linear search, and if found, returns a pointer to this node. If a node with key k is not present in the list, or the list is empty, then the procedure returns the NULL pointer.
- LIST_INSERT_FRONT(L, x) - inserts node x to the front of L.
- LIST_INSERT_TAIL(L, x) - inserts x as the last node of L.
- LIST_INSERT_AFTER(L, x, y) - inserts node x after node y in L.
  Hint: First invoke LIST-SEARCH() function to locate the node y.
- LIST_INSERT_BEFORE(L, x, y) - inserts node x before node y in L.
- LIST_DELETE(L, x) - deletes node x from L.
  Hint: First invoke LIST_SEARCH() function to locate the node x.
- LIST_DELETE_INITIAL(L) - deletes the first node from L and print the deleted node's key.
- LIST_DELETE_LAST(L) - deletes the last node from L and print the deleted node's key..
- PRINT_REVERSE(L, k) - finds the node x with key k and prints all the keys of nodes from x to the head of L, in that order.

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

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 separated by a space. In this operation, the node with this integer as key is inserted into the front of L by calling the function LIST_INSERT_FRONT(L, x).
- Character 't' is followed by an integer separated by a space. In this operation, the node with this integer as key is inserted into the tail of L by calling the function LIST_INSERT_TAIL(L, x).
- Character 'a' is followed by two integers separated by a space. In this operation, the node with the first integer as key is inserted after the node with second integer as key into L by calling the function LIST_INSERT_AFTER(L, x, y).
- Character 'b' is followed by two integers separated by a space. In this operation, the node with the first integer as key is inserted before the node with second integer as key into L by calling the function LIST_INSERT_BEFORE(L, x, y).
- Character 'd' is followed by an integer separated by a space. In this operation, the node with this integer as key is deleted from L and print the key of the node next to the deleted node. Print "NULL" without the quotes, if the deleted node was the last node of L. You can call the function LIST_DELETE(L, x) 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_INITIAL(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 separated by a space. This operation finds the node x with the given integer as key and prints all the keys from x to the head of L, in that order by calling the function PRINT_REVERSE(L, k).
- 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 'd', 'i' , 'l' and 'r', if a node with the input key is not present in L or if L is empty, print "Not Found", without the quotes.

Sample Input

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

Sample Output

38 22 35 20
NULL
20
22
Not Found
Not Found