National Institute of Technology Calicut
Department of Computer Science and Engineering
Third Semester B. Tech.(CSE)
CS2092D Programming Laboratory
Assignment #6

Submission deadline (on or before):

- 21.10.2020, 10:00 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 in the Linux platform.
- 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: *ASSG*1_*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: *ASSG*1_*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 at: http://cse.nitc.ac.in/sites/default/files/Academic-Integrity_new.pdf.

General Instructions

- Programs should be written in C language and compiled using C compiler in Linux platform. **Submit the programs for questions 1 and 2 through the submission link in Eduserver. You should complete the program for question 3 before the next lab session. During the evaluation we may be asking you to modify any of these three programs**.

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. Your program must contain the following functions:

   (In the function prototypes, L, k, x and y denote a Singly Linked List, an integer and 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', 's' 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 pointer *next* of node points to NULL. This procedure returns a pointer to the new node.
   - 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: 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 the 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.
   - LIST-DELETE-LAST(L) - deletes the last node from L.
   - LIST-SEARCH(L, k) - searches for a node with key k in L by doing 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 if the list is empty, the procedure returns NIL.

   Note:- For every INSERT operation, the node x is created by calling CREATE-NODE() function.

   Input format:

   - Each line contains a character from 'f', 't', 'a', 'b', 'd', 'i', 'l', 's' 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 space. In this operation, the node with this integer as key is inserted to the front of L.
   - Character 't' is followed by an integer separated by space. In this operation, the node with this integer as key is inserted to the tail of L.
   - Character 'a' is followed by two integers separated by space. In this operation, the node with the first integer as key is inserted after the node with second integer as key.
   - Character 'b' is followed by two integers separated by space. In this operation, the node with the first integer as key is inserted before the node with second integer as key.
   - Character 'd' is followed by an integer separated by space. In this operation, the node with this integer as key is deleted from L and the deleted node's key is printed.
   - Character 'i' is to delete the first node from L and print the deleted node's key.
   - Character 'l' is to delete the last node from L and print the deleted node's key.
   - Character *'s'* is followed by an integer separated by space. This operation is to find the node with this integer as key in 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' and 'l' , print the deleted node's key. If a node with the input key is not present in L or L is empty, then print -1.
- For option 's', if the key is present in L, then print 1. If key is not present in L or L is empty, then print -1.

Sample Input :

```
f 7
t 10
a 11 7
b 12 11
d 10
i
l
s 12
s 6
e
```

Sample Output:

```
10
7
11
1
-1
```

2. A DOUBLY LINKED LIST L is a data structure in which the objects are arranged in a linear order. Each node of a DOUBLY LINKED LIST 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', 's' 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* of node points to NULL. This procedure returns a pointer to the new node.
   - 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.
     Hint: 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-INITIAL(L) - deletes the first node from L.

- LIST-DELETE-LAST(L) - deletes the last node from L.
- 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 NIL.

Note:- For every INSERT operation, the node x is created by calling CREATE-NODE() function.

Input format:

- Each line contains a character from 'f', 't', 'a', 'b', 'd', 'i', 'l', 's' 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 space. In this operation, the node with this integer as key is inserted into the front of L.
- Character 't' is followed by an integer separated by space. In this operation, the node with this integer as key is inserted into the tail of L.
- Character 'a' is followed by two integers separated by space. In this operation, the node with the first integer as key is inserted after the node with second integer as key into L.
- Character 'b' is followed by two integers separated by space. In this operation, the node with the first integer as key is inserted before the node with second integer as key into L.
- Character 'd' is followed by an integer separated by space. In this operation, the node with this integer as key is deleted from L and the deleted node's key is printed.
- Character 'i' is to delete the first node from L and print the deleted node's key.
- Character 'l' is to delete the last node from L and print the deleted node's key.
- Character *'s'* is followed by an integer separated by space. This operation is to find the node with this integer as key in 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' and 'l', print the deleted node's key. If a node with the input key is not present in L or L is empty, then print -1.
- For option 's', if the key is present in L, then print 1. If key is not present in L or L is empty, then print -1.

Sample Input

```
f 20
t 38
a 22 20
b 35 22
d 38
i
l
s 35
s 40
e
```

Sample Output

```
38
20
22
1
-1
```

3. Operating Systems maintain a data structure called Process Control Block(PCB) for each process. The PCB of a process contains information related to that process. For simplicity, we assume that only *process_id* and *state* of a process are stored in it's PCB and the *process_id*s are distinct. A process can be in any one of the following states: '*new*', '*ready*', '*running*', '*waiting*', or '*terminated*'.

   Write a menu driven program to maintain PCBs of different processes as a SINGLY LINKED LIST called PCB List L. Each PCB of a PCB List L is an object which stores the attributes of a process in the order *process_id* and *state*. Your program must contain the following functions: (In function prototypes, L, k and x denote the PCB List, an integer and a node of L respectively).

   - MAIN() - repeatedly reads a character 'i', 'd', 'g', 'l', 'u' or 'e' from terminal and calls the sub-functions appropriately until character 'e' is entered.
   - CREATE-PCB(k)- Creates a new PCB node with *key* k and the pointer *next* of PCB node points to NULL. This procedure returns a pointer to the new PCB node.
   - INSERT(L, x) - inserts x to the front of L.
   - DELETE(L, x) - deletes x from L.
   - GET-STATE(k, L) - returns the state of process with *process_id*=k in L.
   - LIST-WAITING-PROCESSES(L) - lists the *process_id*s of all processes that are in the state *'waiting'* in L.
   - UPDATE-STATE(L, k, s) - change the state of a PCB with *process_id*=k to s.
   - LIST-SEARCH(L, k) - does a linear search to find the PCB with *process_id* = k in L, and if found, returns a pointer to the located PCB. If such a PCB is not found, or if the list is empty, then the procedure returns NIL.

   Note:- For INSERT operation, the PCB x is created by calling CREATE-PCB() function.

   Input format:

   - Each line contains a character from 'i', 'd', 'g', 'l', 'u', 'e' followed by at most one integer and/or a string. The integers, if given, are in the range $[1, 10^6]$ and the string is from the set {*new*, *ready*, *waiting*, *running*, *terminated*}.
   - Character 'i' is followed by an integer and a string separated by space. In this operation, the PCB with the integer as *process_id* and the string as *state* is inserted into the front of L.
   - Character 'd' is followed by an integer separated by space. In this operation, the PCB with this integer as *process_id* is deleted from L and the deleted PCB's *process_id* is printed (you should first invoke a LIST-SEARCH function to locate the node x with the given *process_id* and pass the pointer to node x to DELETE function).
   - Character 'g' is followed by an integer separated by space. In this operation, the state of PCB with this integer as *process_id* is printed.
   - Character 'l' is to print the *process_id*s of all PCBs having the state as *'waiting'*.
   - Character 'u' followed by an integer k and a string s is to update the state of a PCB with *process_id*=k to the state s.
   - 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 option 'd', print the deleted PCB's *process_id*. If PCB with the given *process_id* is not present in L or if L is empty, then print -1.
   - For option 'g', print the state of PCB with *process_id*=k. If PCB with the given *process_id* is not present in L or if L is empty, then print -1.

- For option '*l*', list the *process_id*s of all PCBs with *state* = '*waiting*' in L, separated by space. If PCB with *state* = '*waiting*' is not present in L, or if L is empty, then print -1.

Sample Input :

```
i 17 new
i 15 new
i 21 new
i 32 new
d 15
u 17 ready
u 32 ready
u 21 ready
u 21 waiting
u 32 waiting
l
g 17
g 76
e
```

Sample Output:

```
15
32 21
ready
-1
```