Data Structure Algorithm

LAB CAT 1

Vaibhav Agrawal

19BIT0185

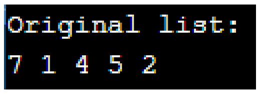
2. Implement doubly linked list by tracing the first and last node and also sort the list in the ascending order. 10 nodes are to be added, 5 nodes to be deleted. Addition and deletion of node in the beginning, middle and end has to be performed. Low Level: Implement any one of the technique [6 marks] Middle Level: Implement the problem by using both the techniques [2 marks] High Level: Implement the above problem to delete every nth node till the list is left with a single node. (n need to be collected as input)

Solution:-

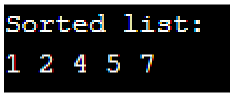
```
#include <stdio.h>
//Represent a node of the doubly linked list
struct node{
   int data;
   struct node *previous;
   struct node *next;
};
//Represent the head and tail of the doubly linked list
struct node *head, *tail = NULL;
//addNode() will add a node to the list
void addNode(int data) {
   //Create a new node
   struct node *newNode = (struct node*)malloc(sizeof(struct node));
```

```
    newNode->data = data;
    //If list is empty
    if(head == NULL) {
        //Both head and tail will point to newNode
        head = tail = newNode;
        //head's previous will point to NULL
        head->previous = NULL;
        //tail's next will point to NULL, as it is the last node of the list
        tail->next = NULL;
    }
    else {
        //newNode will be added after tail such that tail's next will point to newNode
        tail->next = newNode;
        //newNode's previous will point to tail
        newNode->previous = tail;
        //newNode will become new tail
        tail = newNode;
        //As it is last node, tail's next will point to NULL
        tail->next = NULL;
    }
}
//sortList() will sort the given list in ascending order
void sortList() {
    struct node *current = NULL, *index = NULL;
    int temp;
    //Check whether list is empty
    if(head == NULL) {
        return;
    }
    else {
        //Current will point to head
        for(current = head; current->next != NULL; current = current->next) {
            //Index will point to node next to current
            for(index = current->next; index != NULL; index = index->next) {
                //If current's data is greater than index's data, swap the data of current and index
                if(current->data > index->data) {
                    temp = current->data;
                    current->data = index->data;
                    index->data = temp;
                }
            }
        }
    }
}
//display() will print out the nodes of the list
void display() {
    //Node current will point to head
    struct node *current = head;
    if(head == NULL) {
        printf("List is empty\n");
        return;
    }
    while(current != NULL) {
        //Prints each node by incrementing pointer.
        printf("%d ",current->data);
        current = current->next;
    }
    printf("\n");
```

```
}
int main()
{
    //Add nodes to the list
    addNode(7);
    addNode(1);
    addNode(4);
    addNode(5);
    addNode(2);
    //Displaying original list
    printf("Original list: \n");
    display();
    //Sorting list
    sortList();
    //Displaying sorted list
    printf("Sorted list: \n");
    display();
    return 0;
}
```



```
Original list:
7 1 4 5 2
```



```
Sorted list:
1 2 4 5 7
```

1) Add a node at the front:

```
/* Given a reference (pointer to pointer) to the head of a list
   and an int, inserts a new node on the front of the list. */
void push(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

    /* 2. put in the data  */
    new_node->data = new_data;

    /* 3. Make next of new node as head and previous as NULL */
    new_node->next = (*head_ref);
    new_node->prev = NULL;

    /* 4. change prev of head node to new node */
    if ((*head_ref) != NULL)
        (*head_ref)->prev = new_node;

    /* 5. move the head to point to the new node */
    (*head_ref) = new_node;
}
```

2) Add a node at the end:

```
/* Given a reference (pointer to pointer) to the head
   of a DLL and an int, appends a new node at the end  */
void append(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

    struct Node* last = *head_ref; /* used in step 5*/

    /* 2. put in the data  */
    new_node->data = new_data;

    /* 3. This new node is going to be the last node, so
          make next of it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty, then make the new
          node as head */
    if (*head_ref == NULL) {
        new_node->prev = NULL;
        *head_ref = new_node;
        return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;
    /* 6. Change the next of last node */

    last->next = new_node;


    /* 7. Make last node as previous of new node */

    new_node->prev = last;
```

High Level:

```
#include <stdio.h>

#include <stdlib.h>

struct node { // node structure

int data;

struct node *nextpt;
struct node *prevpt;
} *lastpt, *headpt;
void delete_At_Pos(int pos)
{
struct node *extra;
int i,cur;
extra = headpt;
for (i = 1; i < pos && cur != NULL; i++) { extra = extra->nextpt;
}
if (extra != NULL)
{
extra->prevpt->nextpt = extra->nextpt;
extra->nextpt->prevpt = extra->prevpt;
free (extra); // remove the extra node
}
else
{
printf ("Enter valid position!\n");
}
}
void list(int n)
{
```