Practical no: 2

Objective: Write a program in C to create a singly linked-list with following operations:-

- Insertion at beginning.
- Insertion at end
- Insertion at position
- Deletion at begin
- Deletion at end
- Deletion at end

Program Codes: Following is the code of this problems in C:-

1. Practical2a.c

```
#include<stdio.h>
#include<stdlib.h>

// use "node" instead of "struct node" everytime
typedef struct singly_node node;

// globally accessible user-defined data-type
struct singly_node{
    char data;
    struct singly_node * next;
};

// method declarations
// general operations
int node_count(node*);
void display(node*);
// insertion methods declaration
void insert_at_begin(node **);
void insert_at_end(node*);
void insert_at_pos(node*, int);
// deletion method declaration
void del_at_begin(node **);
void del_at_end(node*);
void del_at_pos(node*, int);
```

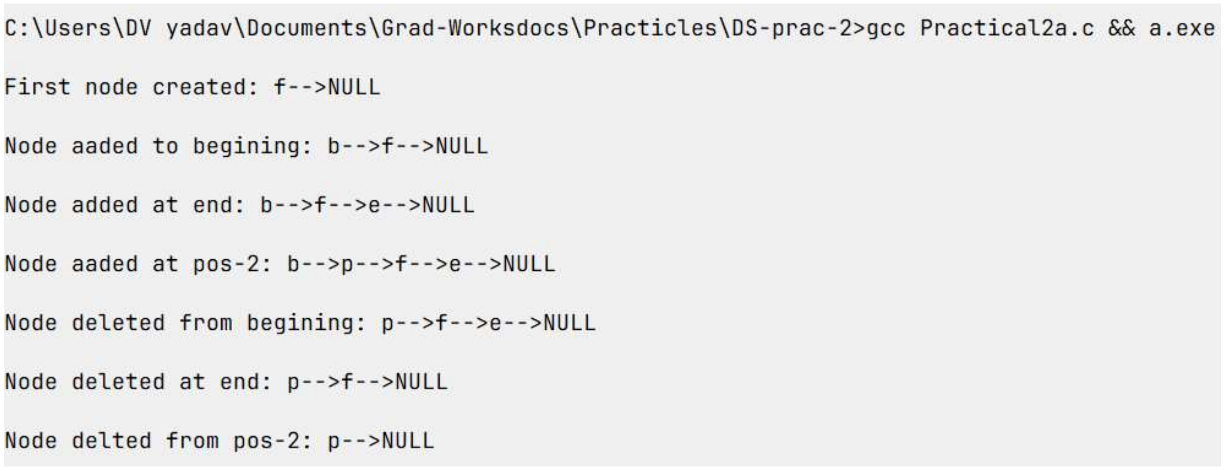
```c
void main(){
    // empty list at first
    node* head = NULL;

    // create first node
    insert_at_begin(&head);
    printf("\nFirst node created: ");
    display(head);

    // insert at beginning of list
    insert_at_begin(&head);
    printf("\nNode added to beginning: ");
    display(head);

    // insert at the end of list
    insert_at_end(head);
    printf("\nNode added at end: ");
    display(head);

    // insert at the position in list
    insert_at_pos(head, 2);
    printf("\nNode added at pos-%d: ", 2);
    display(head);

    // delete at beginning of list
    del_at_begin(&head);
    printf("\nNode deleted from beginning: ");
    display(head);

    // delete at end of list
    del_at_end(head);
    printf("\nNode deleted at end: ");
    display(head);

    // delete at position
    del_at_pos(head, 2);
    printf("\nNode deleted from pos-%d: ", 2);
    display(head);
}
```

```c
// param: address of first node
// return: number of nodes
int node_count(node* temp){
    int count = 0;
    while(temp != NULL){
        count++;
        temp = temp->next;
    }
    return count;
}

// param: address of first node
// prints node data till end
void display(node* temp){
    while(temp != NULL){
        printf("%c--->", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// param: address of the address of first node (head's address)
// uses head pointer via its reference to assigns new node
void insert_at_begin(node** adrs_head){
    // create new node
    node *nptr = (node *)malloc(sizeof(node));
    // if list is empty, create first node
    if(*adrs_head == NULL){
        nptr->data = 'f';
        nptr->next = NULL;
        *adrs_head = nptr;
        return;
    }
    // if not empty, assign new node to head
    nptr->data = 'b';
    nptr->next = *adrs_head;
    *adrs_head = nptr;
}
```

```c
// param: address of first node
// traverse list to end and add new node
void insert_at_end(node* temp){
    // create new node
    node *nptr = (node *)malloc(sizeof(node));
    nptr->data = 'e';
    nptr->next = NULL;

    // traverse till end
    while(temp->next != NULL){
        temp = temp->next;
    }
    // add the new node
    temp->next = nptr;
}

// param: address of first node, position of node insertion
// insertion at first node or at invalid pos. isn't allowed
void insert_at_pos(node* temp, int pos){
    // check for valid positions
    if(pos <= 1 || pos > node_count(temp) ){
        printf("Invalid position: %d\n", pos);
        return;
    }
    // create new node
    node *nptr = (node*)malloc(sizeof(node));
    nptr->data='p';
    // traverse till position
    for(int i = 1; i < pos-1; i++){
        temp = temp->next;
    }
    // insert the node in between
    nptr->next = temp->next;
    temp->next = nptr;
}
```

```c
// param: address of the address of first node (head's address)
// uses head pointer via its reference to delete first node
void del_at_begin(node** adrs_head){
    // de-allocator variable to free node memory
    node *dlctr = * adrs_head;
    *adrs_head = dlctr->next;
    // delete node and pointer
    free(dlctr);
    dlctr = NULL;
}

// param: address of first node
// traverse till end to delete last node
void del_at_end(node * temp){
    // traverse till end
    while(temp->next->next != NULL){
        temp = temp->next;
    }
    // delete node and pointer
    free(temp->next);
    temp->next = NULL;
}

// param: address of first node, position of node deletion
// deletion at first node or at invalid pos. isn't allowed
void del_at_pos(node* temp, int pos){
    // check for valid position
    if(pos <= 1 || pos > node_count(temp)){
        printf("Invalid position: %d\n", pos);
        return;
    }
    // traverse till position
    for(int i = 1; i < pos-1; i++){
        temp = temp->next;
    }
    // de-allocator variable to free node memory
    node* dlctr = temp->next;
    temp->next = dlctr->next;
    // delete node and pointer
    free(dlctr);
    dlctr = NULL;
}
```

Output: Following is the output of the program:-

1. 

```
C:\Users\DV yadav\Documents\Grad-Worksdocs\Practicles\DS-prac-2>gcc Practical2a.c && a.exe

First node created: f-->NULL

Node aaded to begining: b-->f-->NULL

Node added at end: b-->f-->e-->NULL

Node aaded at pos-2: b-->p-->f-->e-->NULL

Node deleted from begining: p-->f-->e-->NULL

Node deleted at end: p-->f-->NULL

Node delted from pos-2: p-->NULL
```