A picture containing logo

Description automatically generated ***East West University***

**Implementation of XOR-linked lists.**

**Course Title :** Data Structures

**Course Code :** CSE207

**Section :** 02

**Semester :** Spring 2021

**Submitted To**

**Dr. Maheen Islam,**

Associate Professor,,

Department of Computer Science and Engineering

**Submitted By**

**Group members:**

Rafiur Rahman Rafit

2018-3-60-111

Jaser Maharus

2018-3-60-070

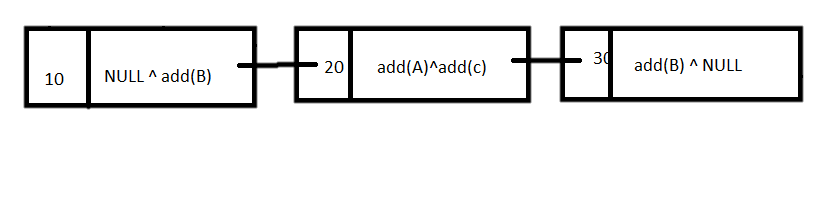
**Submission Date**

June 2,2021

**Implementation of XOR-linked lists**

**Introduction**

XOR Linked Listed Can go forward and backward through it is a singly Linked List. It Worked Liked Doubly Linked List



Here some used function,

* insert\_at\_first(int value)
* insert\_at\_last(int value)
* int delete\_from\_fast()
* int delete\_from\_last()
* display()
* size\_list().

The Function given bellow

int data;

struct Node\* npx;

// nps of next node and previous node

};

// Method 1

// It returns Xored value of the node addresses

struct Node \*head, \*tail;

struct Node\* Xor(struct Node \*a,struct Node \*b){

return (struct Node\*) ((uintptr\_t) (a) ^ (uintptr\_t) (b));

}

void insert\_at\_first(int value)

{ //function to enter node at First//

struct Node \*node =(struct Node\*)

malloc(sizeof(struct Node));; // Allocate memory for new node

node->data = value;

if (head==NULL) {

node->npx = NULL;

head = tail = node;

}

else {

node->npx = Xor(NULL, head);

head->npx = Xor(node, head->npx);

head = node;

}

}

void insert\_at\_last(int value) //function to enter node at last//

{

struct Node \*node =(struct Node\*)

malloc(sizeof(struct Node));

node->data = value;

if (head==NULL) {

node->npx = NULL;

head = tail = node;

}

else {

node->npx = Xor(tail, NULL);

tail->npx = Xor(node, tail->npx);

tail = node;

}

}

int delete\_from\_fast()

{ // Function to remove the first node form

int item;

struct Node \*ptr;

if (head==NULL) {

printf("Empty list.\n");

}

else {  // Store the node to be deleted

ptr = head;

item = ptr->data; //value update//

struct Node \*next = Xor(NULL, ptr->npx); // Update head//

if (NULL == next)

{

tail = NULL;

}

else

next->npx = Xor(ptr, next->npx);

head = next;

}

free(ptr);

ptr = NULL;

return item;

}

int delete\_from\_last()

{ // Function to remove the last node form

int item;

struct Node \*ptr;

if (head==NULL) {

printf("Empty list.\n");

}

else {

// Update previous

ptr = tail;

  // Update item

item = ptr->data;

struct Node \*prev = Xor(ptr->npx, NULL);

if (NULL == prev){

head = NULL;

}

else

prev->npx= Xor(ptr, prev->npx);

tail = prev;

}

  // Delete the last node from memory

free(ptr);

ptr = NULL;

return item;

}

void display()

{

 // Stores XOR pointer

    // in current node

struct Node \*curr = head;

struct Node \*prev = NULL,  // Stores XOR pointer of

    // in previous Node

\*next;

printf("\nList elements are : ");

while (curr!=NULL) {   // Traverse XOR linked list

printf("\n%d ",curr->data);

  // Forward traversal

next = Xor(prev, curr->npx);

prev = curr;   // Update prev

curr = next;  // Update curr

}

printf("\n ");

}

void size\_list()

{

int c=0;

struct Node \*curr = head;

struct Node \*prev = NULL, \*next;

while (curr!=NULL) {

c++;

next = Xor(prev, curr->npx);

prev = curr;

curr = next;

}

printf("Size is %d\n",c);

}

**Requirements to run the program:**

* Code blocks or any c compiler
* Operating System
* Windows/Linux distribution(kali/ubuntu/Parrot/mint)/Mac

**insert\_at\_first(int value)**

this function used to insert a node at the beginning of the XOR linked list and makes the newly inserted node as head.

**insert\_at\_last(int value)**

this function used to insert a node at the End of the XOR linked list and makes the newly inserted node as head.

**int delete\_from\_fast()**

This function used to delete node at beginning and update head pointer and address.

**int delete\_from\_last()**

This function used to delete node at end and update previous node and address.

**Benefits:**

* It Work Like Doubly Linked List
* It is memory efficient than doubly linked List
* Instead of storing next node, it store XOR node

**Demerit:**

* Code is hard and complex.
* Debugging hard

**Conclusion**

Though it is hard and complex but it helpful in work because it is singly linked list but works like doubly linked list and it is memory efficient also because it stores XOR address of next and previous in a single node.