/\* Memory efficient Linked List \*/

#include <stdio.h>

#include <stdlib.h>

#include <inttypes.h>

// Node structure

struct Node

{

int data;

struct Node\* npx;

};

/\* Function to find the XOR of two nodes \*/

struct Node\* XOR (struct Node \*a, struct Node \*b)

{

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

}

// Function to insert a node at given position

void insert(struct Node\*\* head\_ref,int value,

int position)

{// If list is empty

if (\*head\_ref == NULL)

{

// If given position = 1

if (position == 1)

{

// Initialize a new Node

struct Node\* node =

(struct Node\*)malloc(sizeof(struct Node));

// Stores data value in the node

node->data = value;

// Stores XOR of previous and next pointer

node->npx = XOR (NULL, NULL);

// Update pointer of head\_ref node

\*head\_ref = node;

printf("Node with data %d inserted at Position %d \n",node->data,position);

}

// If required position was not found

else

{

printf("Invalid Position %d as list is Empty\n", position);

}

}

// If the linked list is not empty

else

{

// Stores position of a node for Tracking the position

int Pos = 1;

// Stores the address of current node

struct Node\* curr = \*head\_ref;

// Stores the address of previous node

struct Node\* prev = NULL;

// Stores the XOR of next node and previous node

struct Node\* next = XOR(prev, curr->npx);

// Traverse the XOR linked list

while (next != NULL && Pos < position - 1)

{

// Update prev

prev = curr;

// Update curr

curr = next;

// Update next

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

// Update Pos

Pos++;

}

// If the position of the current node is equal to the given position

if (Pos == position -1)

{

// Initialize a new Node

struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));

// Stores pointer to previous Node as (prev ^ next ^ next) = prev

struct Node\* temp = XOR(curr->npx, next);

// Stores XOR of prev and new node

curr->npx = XOR(temp, node);

// Connecting new node with next

if (next != NULL)

{

// Update pointer of next

next->npx = XOR(node,XOR(next->npx, curr));

}

// Connect node with curr and next curr<--node-->next

node->npx = XOR(curr, next);

node->data = value;

printf("Node with data %d inserted at Position %d\n",node->data,position);

}

// Insertion node at beginning

else if (position == 1)

{

// Initialize a new Node

struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));

// Update curr node address

curr->npx = XOR(node,XOR(NULL, curr->npx));

// Update new node address

node->npx = XOR(NULL, curr);

// Update head\_ref

\*head\_ref = node;

// Update data value of current node

node->data = value;

printf("Node with data %d inserted at Position %d \n",node->data,Pos);

}

else

{

printf("Invalid Position %d\n",position);

}

}

}

int delEnd (struct Node \*\*head,int pos)

{

int item=0;

if (\*head == NULL)

printf ("List is empty");//if the list doesn't contain any node

else

{

int flag=0;

// Stores XOR pointer

// in current node

struct Node \*curr = \*head;

// Stores XOR pointer of

// in previous Node

struct Node \*prev = NULL;

// Stores XOR pointer of

// in next node

struct Node \*next;

// Traverse XOR linked list

while (XOR (curr->npx, prev) != NULL && flag<pos)

{

// Forward traversal

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

// Update prev

prev = curr;

// Update curr

curr = next;

flag++;

}

printf("%d\n",flag);

// If the Linked List contains more than 1 node

if(flag==pos){

item=curr->data;

if (prev != NULL)

prev->npx = XOR (XOR (prev->npx, curr), NULL);

// Otherwise

else

\*head = NULL;

// Delete the last node from memory

free (curr);

}

else

printf("Invalid position");}

// Returns head of new linked list

return item;

}

// prints contents of doubly linked list

void printList (struct Node \*head\_ref)

{

struct Node \*curr = head\_ref;

struct Node \*prev = NULL;

if(head\_ref==NULL)

printf("Empty List\n");

else{

struct Node \*next;

while (curr != NULL)

{

// print current node

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

// get address of next node: curr->npx is next^prev, so curr->npx^prev will be next^prev^prev which is next

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

// update prev and curr for next iteration

prev = curr;

curr = next;

}}

printf("\n");

}

void traverseback(struct Node\*\* head\_ref)

{

struct Node\* curr = \*head\_ref;

struct Node\* prev = NULL;

struct Node\* next = XOR(prev, curr->npx);

if(curr!=NULL)

{

while (XOR(prev,curr->npx) !=NULL)

{

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

prev=curr;

curr=next;

}

\*head\_ref=curr;

}

}

int main ()

{

struct Node \*head = NULL;

int ch,ele,pos;

while(1)

{

printf("1.Enter 1. to insert\n2.Enter 2. to delete\n3.Enter 3. for forward traversal\n4.Enter 4. for backward traversal\nPress any other key to exit\n");

printf("Enter your choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter element and the position at which it is to be inserted\n");

scanf("%d%d",&ele,&pos);

insert(&head,ele,pos);

break;

case 2:

if(head==NULL)

printf("List is empty\n");

else{

printf("Enter position at which element is to be deleted\n");

scanf("%d",&pos);

printf("Deleted element is:%d\n",delEnd(&head,pos));}

break;

case 3:

printf("Forward Traversal of List is\n");

printList(head);

break;

case 4:

printf("Backward Traversal of List is\n");

if(head==NULL)

printf("Empty List\n");

else

{

traverseback(&head);

printList(head);

traverseback(&head);

}

break;

default:

printf("Exiting program");

exit(0);}}

return (0);

}

/\* Memory efficient Linked List \*/

#include <stdio.h>

#include <stdlib.h>

#include <inttypes.h>

// Node structure

struct Node

{

int data;

struct Node\* npx;

};

/\* Function to find the XOR of two nodes \*/

struct Node\* XOR (struct Node \*a, struct Node \*b)

{

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

}

int c=0;

// Function to insert a node at given position

void insert(struct Node\*\* head\_ref,int value,

int position)

{

// If list is empty

if (\*head\_ref == NULL)

{

// If given position = 1

if (position == 1)

{

// Initialize a new Node

struct Node\* node =

(struct Node\*)malloc(sizeof(struct Node));

// Stores data value in the node

node->data = value;

// Stores XOR of previous and next pointer

node->npx = XOR (NULL, NULL);

// Update pointer of head\_ref node

\*head\_ref = node;

printf("Node with data %d inserted at Position %d \n",node->data,position);

}

// If required position was not found

else

{

printf("Invalid Position %d as list is Empty\n", position);

}

}

// If the linked list is not empty

else

{

// Stores position of a node for Tracking the position

int Pos = 1;

// Stores the address of current node

struct Node\* curr = \*head\_ref;

// Stores the address of previous node

struct Node\* prev = NULL;

// Stores the XOR of next node and previous node

struct Node\* next = XOR(prev, curr->npx);

// Traverse the XOR linked list

while (next != NULL && Pos < position - 1)

{

// Update prev

prev = curr;

// Update curr

curr = next;

// Update next

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

// Update Pos

Pos++;

}

// If the position of the current node is equal to the given position

if (Pos == position -1)

{

// Initialize a new Node

struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));

// Stores pointer to previous Node as (prev ^ next ^ next) = prev

struct Node\* temp = XOR(curr->npx, next);

// Stores XOR of prev and new node

curr->npx = XOR(temp, node);

// Connecting new node with next

if (next != NULL)

{

// Update pointer of next

next->npx = XOR(node,XOR(next->npx, curr));

}

// Connect node with curr and next curr<--node-->next

node->npx = XOR(curr, next);

node->data = value;

printf("Node with data %d inserted at Position %d\n",node->data,position);

}

// Insertion node at beginning

else if (position == 1)

{

// Initialize a new Node

struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));

// Update curr node address

curr->npx = XOR(node,XOR(NULL, curr->npx));

// Update new node address

node->npx = XOR(NULL, curr);

// Update head\_ref

\*head\_ref = node;

// Update data value of current node

node->data = value;

printf("Node with data %d inserted at Position %d \n",node->data,Pos);

}

else

{

printf("Invalid Position %d\n",position);

}

}

}

int delEnd(struct Node \*\*head,int pos)

{

c=0;

int item;

struct Node\* temp1;

if (\*head == NULL)

printf ("List is empty");//if the list doesn't contain any node

else

{

int flag=1;

struct Node \*curr = \*head;

struct Node \*prev = NULL;

struct Node \*next=XOR (prev, curr->npx);

while (XOR (prev,curr->npx ) != NULL && flag<pos-1)

{

prev = curr;

curr = next;

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

flag++;

}

if(flag==pos-1){

temp1=XOR(prev,curr->npx);

struct Node\* temp2=XOR(curr,temp1->npx);

if(temp2!=NULL){

struct Node\* temp3=XOR(temp1,temp2->npx);

curr->npx=XOR(prev,temp2);

temp2->npx=XOR(curr,temp3);

item=temp1->data;

}

else if(temp1!=NULL){

curr->npx=XOR(prev,NULL);

item=temp1->data;

free(temp1);

}

return item;}

// Otherwise

else if(pos==1){

item=(\*head)->data;

\*head=XOR(NULL,curr->npx);

// Delete the last node from memory

if(\*head!=NULL){

temp1=XOR(NULL,curr->npx);

struct Node\* temp2=XOR(curr,temp1->npx);

if(temp1!=NULL){

temp1->npx=XOR(NULL,temp2);

\*head=temp1;}}

free(curr);

return item;}

c++;

}}

// Returns head of new linked list

// prints contents of doubly linked list

void printList (struct Node \*head\_ref)

{

struct Node \*curr = head\_ref;

struct Node \*prev = NULL;

if(head\_ref==NULL)

printf("Empty List\n");

else{

struct Node \*next;

while (curr != NULL)

{

// print current node

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

// get address of next node: curr->npx is next^prev, so curr->npx^prev will be next^prev^prev which is next

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

// update prev and curr for next iteration

prev = curr;

curr = next;

}}

printf("\n");

}

void traverseback(struct Node\*\* head\_ref)

{

struct Node\* curr = \*head\_ref;

struct Node\* prev = NULL;

struct Node\* next = XOR(prev, curr->npx);

if(curr!=NULL)

{

while (XOR(prev,curr->npx) !=NULL)

{

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

prev=curr;

curr=next;

}

\*head\_ref=curr;

}

}

int main ()

{

struct Node \*head = NULL;

int ch,ele,pos,res,a;

while(1)

{

printf("1.Enter 1. to insert\n2.Enter 2. to delete\n3.Enter 3. for forward traversal\n4.Enter 4. for backward traversal\nPress any other key to exit\n");

printf("Enter your choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter element and the position at which it is to be inserted\n");

scanf("%d%d",&ele,&pos);

insert(&head,ele,pos);

break;

case 2:

if(head==NULL)

printf("List is empty\n");

else{

printf("Enter position at which element is to be deleted\n");

scanf("%d",&pos);

res=delEnd(&head,pos);

if(c!=0)

printf("Invalid position\n");

else

printf("Deleted element is:%d\n",res);}

break;

case 3:

printf("Forward Traversal of List is\n");

printList(head);

break;

case 4:

printf("Backward Traversal of List is\n");

if(head==NULL)

printf("Empty List\n");

else

{

traverseback(&head);

printList(head);

traverseback(&head);

}

break;

default:

printf("Exiting program");

exit(0);}}

return (0);

}