Experiment No: 8 Date: 30-Nov-2021

Aim:

To implement operations of a doubly linked list

Algorithm:

1.START

2.Display a menu of operations

3. If choice is for insertion

3.1. If beginning

3.1.1.set previous node of newnode to NULL

3.1.2.set next node of newnode to head

3.1.3.set head to point to newnode

3.2. If end

3.2.1.set temp to point to the head (temp=head)

3.2.2.Travel the doubly linked list till the next of temp is null

3.2.3.insert newnode into temp->next

3.2.4.set newnode->next as NULL

3.2.5.set newnode->previous=temp

3.3. If particular position

3.3.1.set temp to point to the first node (head)

3.3.2.travel till the desired position reach

3.3.3.insert the newnode at next to the temp

3.3.4.set previous pointer of newnode to temp

3.3.5.set next pointer of newnode to the current next of temp

3.3.6.set next pointer of temp to newnode

4. If choice is for deletion

4.1. If beginning

4.1.1.make temp point to first node

4.1.2.set previous link of next link of temp to be NULL

4.1.3. Set head pointing to the node next to temp;

4.1.4..set NULL at next pointer of temp

4.2. If end

4.2.1.assign temp as newnode

4.2.2.traverse the list till next of temp equal to NULL

4.2.3.set next pointer of previous link of temp to NULL

4.2.4.set previous pointer of temp to NULL, and free temp

4.3. If specific position

4.3.1. Set temp pointing to the first node (head)

4.3.2. Read a logical position from which the node is to be removed

4.3.3. Remove the desired node by traversing the list

4.3.4. Set the next pointer of node residing before temp as the node after

temp

4.3.5. Set previous link of node after temp pointing to node before temp

5. If choice is for searching an item in the list, traverse the list by checking that

whethether the item is matching with the data part of current node

5.1. Assign temp = head

5.2. while(temp->data!=item),traverse by temp=temp->next

5.3. If data in temp is equal to item, print ‘ item found in list’

5.4. Otherwise print “ item does not exist”

6. If operation is for traversal or display

6.1. Set temp as head

6.2. Print the data contained in temp while temp reaches last node

7. If user’s choice is none other than above , print ‘invalid choice’

8. Continue steps 2 to 7 till the user input an option for exit

9. STOP

Program:-

#include<stdio.h>

#include<stdlib.h>

int count=0;

void insert\_begin();

void insert\_end();

void insert\_pos();

void delete\_begin();

void delete\_end();

void delete\_pos();

void search\_key();

void traverse\_list();

struct node

{

int data;

struct node \*prev;

struct node \*next;

}\*head=NULL;

void main()

{

int opt,item;

do

{

printf("\n SELECT A VALID OPTION FROM THE MENU\n");

printf("\n1. INSERTION AT BEGINNING\n");

printf("\n2. INSERTION AT END\n");

printf("\n3. INSERTION AT A GIVEN POSITION\n");

printf("\n4. DELETION AT BEGINNING\n");

printf("\n5. DELETION AT END\n");

printf("\n6. DELETION AT A PARTICULAR POSITION\n");

printf("\n7. SEARCH FOR AN ITEM\n");

printf("\n8. DISPLAY LIST\n");

printf("\n9. EXIT\n");

scanf("%d",&opt);

switch(opt)

{

case 1: insert\_begin();

break;

case 2: insert\_end();

break;

case 3: insert\_pos();

break;

case 4: delete\_begin();

break;

case 5: delete\_end();

break;

case 6: delete\_pos();

break;

case 7: search\_key();

break;

case 8: traverse\_list();

break;

case 9: exit(0);

default: printf("\n Invalid Option\n");

}

}

while(opt!=9);

}

void insert\_begin()

{

int item;

printf("\n enter a value: ");

scanf("%d",&item);

struct node \*newnode;

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

newnode->data=item;

if(head==NULL)

{

head=newnode;

newnode->prev=NULL;

newnode->next=NULL;

count++;

}

else

{

struct node \*temp=head;

temp->prev=newnode;

newnode->prev=NULL;

newnode->next=temp;

head=newnode;

count++;

}

printf("\n the items in the list are:\n");

traverse\_list();

}

void insert\_end()

{

int item;

printf("\n enter a value: ");

scanf("%d",&item);

struct node \*newnode;

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

newnode->data=item;

if(head==NULL)

{

head=newnode;

newnode->prev=NULL;

newnode->next=NULL;

count++;

}

else

{

struct node \*temp=head;

while(temp->next!=NULL)

temp=temp->next;

temp->next=newnode;

newnode->prev=temp;

newnode->next=NULL;

count++;

}

printf("\n the items in the list are\n");

traverse\_list();

}

void insert\_pos()

{

int item,pos,i=1;

struct node \*temp=head;

printf("\n enter a value: ");

scanf("%d",&item);

struct node \*newnode;

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

newnode->data=item;

printf("\n Enter the position to which the new node is to be inserted: ");

scanf("%d",&pos);

if(pos>count)

{

printf("\n invalid position\n");

}

while(temp->next!=NULL&&i!=pos-1)

{

temp=temp->next;

i++;

}

if(i==pos-1)

{

newnode->next=temp->next;

temp->next=newnode;

newnode->prev=temp;

count++;

}

else

{

if(pos==count)

{

while(temp->next!=NULL)

temp=temp->next;

temp->next=newnode;

newnode->next=NULL;

newnode->prev=temp;

count++;

}

else

printf("\n POSITION not found in list\n");

}

printf("\n the items in the list are\n");

traverse\_list();

}

void delete\_begin()

{

struct node \*temp=head;

if(head==NULL)

printf("\n doubly linked list is empty\n");

else

{

if(temp->next==NULL)

{

temp->prev=NULL;

head=NULL;

printf("\n the item %d has been deleted\n",temp->data);

free(temp);

count--;

traverse\_list();

}

else

{

head=temp->next;

temp->next->prev=NULL;

temp->prev=NULL;

temp->next=NULL;

printf("\n the item %d has been deleted from beginning\n",temp->data);

free(temp);

count--;

printf("\n the items in the list are\n");

traverse\_list();

}

}

}

void delete\_end()

{

struct node \*temp=head;

if(head==NULL)

printf("\n doubly linked list is empty\n");

else if(temp->next==NULL)

{

printf("\n the item %d has been deleted\n",temp->data);

temp->prev=NULL;

temp->next=NULL;

head=NULL;

free(temp);

count--;

printf("\n the items in the list are\n");

traverse\_list();

}

else

{

while(temp->next!=NULL)

temp=temp->next;

temp->prev->next=NULL;

temp->prev=NULL;

printf("\n the item %d has been deleted from end\n",temp->data);

free(temp);

count--;

printf("\n the items in the list are\n");

traverse\_list();

}

}

void delete\_pos()

{

int pos,i=1;

struct node \*temp=head;

if(head==NULL)

printf("\n the doubly linked list is empty\n");

else

{

printf("\n enter the position of node to be deleted: ");

scanf("%d",&pos);

if(pos>count)

printf("\n position is not within the list\n");

else

{

while(temp->next!=NULL&&pos!=i)

{

temp=temp->next;

i++;

}

temp->prev->next=temp->next;

temp->prev=NULL;

temp->next=NULL;

printf("\n the item %d has been deleted",temp->data);

free(temp);

count--;

printf("\n the items in the doubly linked list are\n");

traverse\_list();

}

}

}

void traverse\_list()

{

struct node \*temp=head;

if(head==NULL)

printf("\n list is empty\n");

else

{

while(temp!=NULL)

{

printf("%d\t",temp->data);

temp=temp->next;

}

}

}

void search\_key()

{

int item;

printf("\n enter an item to be searched: \n");

scanf("%d",&item);

struct node \*temp=head;

while(temp->data!=item&&temp->next!=NULL)

temp=temp->next;

if(temp->data==item)

printf("\n the item %d found in the list",item);

else

printf("\n the item %d not found in the list\n",item);

}