/\*\*\*\*\*\*DOUBLY LINKED LIST\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

struct node {

int data;

struct node \*prev;

struct node \*next;

};

struct node \*s1=NULL;

struct node \*insert\_begin(struct node \*start)

{

struct node \*temp;

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

printf("Enter the value to be inserted\n");

scanf("%d",&temp->data);

temp->next=NULL;

temp->prev=NULL;

if(start==NULL)

{

start=temp;

}

else

{

temp->next=start;

start->prev=temp;

start=temp;

}

return start;

}

struct node \*insert\_end(struct node \*start)

{

struct node \*temp;

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

printf("Enter the value to be inserted\n");

scanf("%d",&temp->data);

temp->next=NULL;

temp->prev=NULL;

if(start==NULL)

{

start=temp;

}

else{

struct node \*ptr;

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

temp->prev=ptr;

}

return start;

}

struct node \*insert\_pos(struct node \*start)

{

int n;

struct node \*temp;

struct node \*ptr=start;

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

printf("Enter the value to be inserted\n");

scanf("%d",&temp->data);

temp->next=NULL;

temp->prev=NULL;

printf("Enter the position where the node has to be inserted\n");

scanf("%d",&n);

if(n==1)

{

temp->next=start;

start=temp;

}

else

{

for(int i=1;i<n-1;i++)

{

ptr=ptr->next;

if(ptr==NULL)

{

printf("the node cant be inserted at position -%d\n",n);

}

}

temp->next=ptr->next;

ptr->next=temp;

}

return start;

}

struct node \*delete\_begin(struct node \*start)

{

if(start==NULL)

{

printf("Empty list\n");

return start;

}

else if(start->next==NULL)

{

printf("Value deleted=%d",start->data);

free(start);

start=NULL;

}

else{

struct node \*ptr;

ptr=start;

start=start->next;

printf("Value deleted=%d",ptr->data);

free(ptr);

start->prev=NULL;

}

return start;

}

struct node \*delete\_end(struct node\*start)

{

struct node \*ptr=start;

if(start==NULL)

{

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

return start;

}

else if(start->next==NULL)

{

printf("The value deleted=%d",start->data);

free(start);

start=NULL;

}

else{

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

printf("the value deleted=%d",ptr->data);

ptr->prev->next=NULL;

free(ptr);

}

return start;

}

struct node \*delete\_pos(struct node \*start)

{

int n;

struct node \*ptr=start;

if(start==NULL)

{

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

return start;

}

printf("Enter the position to be deleted\n");

scanf("%d",&n);

if(n==1)

{

printf("The value deleted=%d",start->data);

start=start->next;

start->prev=NULL;

free(ptr);

}

else{

for(int i=1;i<n-1;i++)

ptr=ptr->next;

if(ptr==NULL)

{

printf("the node cant be inserted at position -%d\n",n);

}

printf("The value deleted=%d",ptr->next->data);

struct node\* tempr=ptr->next;

ptr->next=ptr->next->next;

free(temp);

ptr->next->prev=ptr;

}

return start;

}

/\*\*\*ITS

/ Function to remove duplicates from a sorted list \*/

void duplicate(struct node \*temp)

{

struct node \*ptr1, \*ptr2, \*duplicate;

ptr1 = temp;

while (ptr1 != NULL && ptr1->next != NULL)

{

ptr2 = ptr1;

/\* Compare the current element with rest of the elements \*/

while (ptr2->next != NULL)

{

if (ptr1->data == ptr2->next->data)

{

duplicate = ptr2->next;

ptr2->next = ptr2->next->next;

free(duplicate);

}

else

ptr2 = ptr2->next;

}

ptr1 = ptr1->next;

}

}

void display\_reverse(struct node \*start)

{

struct node \*ptr,\*pvv;

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

pvv=ptr;

//remember its pvv!=NULL if u put pvv->next!=null the last element wont get printed

while(pvv !=NULL)

{

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

pvv=pvv->prev;

}

}

void display(struct node \*start)

{

struct node \*ptr;

ptr=start;

if(start==NULL)

{

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

}

else{

while(ptr!=NULL)

{

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

ptr=ptr->next;

}

}

}

int main()

{

int choice;

while(1)

{

printf("\n 1. to add in the beginning \n 2. to add at the end\n 3. to insert at the given position\n 4.to delete at the beginning\n 5.to delete at the end\n 6.to delete at a specific position\n 7. to delete the duplicate elements\n 8.display 9. to display the list in reverse order \n");

scanf("%d",&choice);

switch(choice)

{

case 1: s1=insert\_begin(s1);

break;

case 2: s1=insert\_end(s1);

break;

case 3:s1=insert\_pos(s1);

break;

case 4:s1=delete\_begin(s1);

break;

case 5:s1=delete\_end(s1);

break;

case 6: s1=delete\_pos(s1);

break;

case 7: duplicate(s1);

break;

case 8:display(s1);

break;

case 9:display\_reverse(s1);

break;

}

}

}