/\*doubly linked list full op\*/

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\*rlink;

struct node\*llink;

};

struct node\*header;

struct node\*create\_ll(struct node\*);

struct node\*display(struct node\*);

struct node\*insert\_beg(struct node\*);

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

struct node\*insert\_any(struct node\*);

struct node\*delete\_beg(struct node\*);

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

struct node\*delete\_any(struct node\*);

void search();

struct node\*sort\_list(struct node\*);

int main()

{

int choice=0;

while(choice!=11)

{

printf("\*\*main menu\*\*\n");

printf("1.create list\n2.display the list\n3.insert at the begining\n4.insert at the end\n5.insert at any position\n6.delete from the begining\n7.delete from the end\n8.delete from any position\n9.search\n10.sort the list\n11.exit\n");

printf("enter your choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:header=create\_ll(header);

break;

case 2:header=display(header);

break;

case 3:header=insert\_beg(header);

break;

case 4:header=insert\_end(header);

break;

case 5:header=insert\_any(header);

break;

case 6:header=delete\_beg(header);

break;

case 7:delete\_end(header);

break;

case 8:delete\_any(header);

break;

case 9:search();

break;

case 10:header=sort\_list(header);

break;

case 11:exit(0);

default:

printf("invalid choice\n");

}

}

}

struct node\*create\_ll(struct node\*header)

{

struct node\*new\_node,\*ptr;

int item;

printf("enter -1 to end\n");

printf("enter the data: \n");

scanf("%d",&item);

while(item!=-1)

{

if(header==NULL)

{

new\_node=(struct node\*)malloc(sizeof(struct node\*));

new\_node->llink=NULL;

new\_node->data=item;

new\_node->rlink=NULL;

header=new\_node;

}

else

{

ptr=header;

new\_node=(struct node\*)malloc(sizeof(struct node\*));

new\_node->data=item;

while(ptr->rlink!=NULL)

{

ptr=ptr->rlink;

}

ptr->rlink=new\_node;

new\_node->rlink=NULL;

}

printf("enter the data: \n");

scanf("%d",&item);

}

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

return header;

}

struct node\*display(struct node\*header)

{

printf("the linked list is below\n");

struct node\*ptr;

ptr=header;

while(ptr!=NULL) //list is not empty

{

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

ptr=ptr->rlink;

}

return header;

}

struct node\*insert\_beg(struct node\*header)

{

struct node\*new\_node,\*ptr;

int item;

if(header==NULL)

{

printf("overflow:insertion not possible\n"); //memory bank returns NULL

}

else

{

printf("enter the data to be inserted: \n");

scanf("%d",&item);

new\_node=(struct node\*)malloc(sizeof(struct node\*));

new\_node->data=item;

header->llink=new\_node;

new\_node->rlink=header;

new\_node->llink=NULL;

header=new\_node;

printf("node inserted at the begining\n");

return header;

}

/\*other process for else part

{

printf("enter the data to be inserted: \n"); {here,

scanf("%d",&item); header is not a node in ll}

new\_node=(struct node\*)malloc(sizeof(struct node\*));

new\_node->data=item;

ptr=header->link;

new\_node->rlink=ptr;

new\_node->llink=header;

header->rlink=new\_node;

ptr->llink=new\_node;

}

\*/

}

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

{

struct node\*new\_node,\*ptr;

int item;

if(header==NULL)

{

printf("overflow:insertion not possible\n"); //memory bank returns NULL

}

else

{

printf("enter the data to be inserted: \n");

scanf("%d",&item);

new\_node=(struct node\*)malloc(sizeof(struct node\*));

new\_node->data=item;

ptr=header;

while(ptr->rlink!=NULL)

{

ptr=ptr->rlink;

}

ptr->rlink=new\_node;

new\_node->llink=ptr;

new\_node->rlink=NULL;

printf("node inserted at the end\n");

return header;

}

}

struct node\*insert\_any(struct node\*header)

{

struct node\*new\_node,\*ptr;

int loc,i,item;

if(header==NULL)

{

printf("overflow:insertion not possible\n"); //memory bank returns NULL

}

else

{

printf("enter the location after which the node has to be inserted\n");

scanf("%d",&loc);

printf("enter the data to be inserted: \n");

scanf("%d",&item);

new\_node=(struct node\*)malloc(sizeof(struct node\*));

new\_node->data=item;

ptr=header;

for(i=0;i<loc;i++) //the linked list is started from 0th index here

{

ptr=ptr->rlink;

}

new\_node->llink=ptr;

new\_node->rlink=ptr->rlink;

ptr->rlink->llink=new\_node;

ptr->rlink=new\_node;

printf("node inserted at specific position\n");

return header;

}

}

struct node\*delete\_beg(struct node\*header)

{

struct node\*ptr;

if(header==NULL)

{

printf("deletion not possible\n"); //list is empty

}

else

{

ptr=header;

header=header->rlink;

header->llink=NULL;

free(ptr);

printf("node is deleted from the begining\n");

return header;

}

}

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

{

struct node\*ptr,\*ptr1;

if(header==NULL)

{

printf("deletion not possible\n"); //list is empty

}

else

{

ptr=header;

while(ptr->rlink!=NULL)

{

ptr1=ptr;

ptr=ptr->rlink;

}

ptr1->rlink=NULL;

free(ptr);

printf("node is deleted from the end\n");

return header;

}

}

struct node\*delete\_any(struct node\*header)

{

struct node\*ptr,\*ptr1;

int i,loc,item;

if(header==NULL)

{

printf("deletion not possible\n"); //list is empty

}

else

{

printf("enter the location after which the node has to be deleted\n");

scanf("%d",&loc);

ptr=header;

for(i=0;i<=loc;i++) //the linked list is started from 0th index here

{

ptr1=ptr;

ptr=ptr->rlink;

}

ptr->rlink->llink=ptr1;

ptr1->rlink=ptr->rlink;

free(ptr);

printf("node deleyed from specific position\n");

return header;

}

}

void search()

{

struct node\*ptr;

int item,i=0,flag=0,loc;

if(header==NULL)

{

printf("empty list\n");

}

else

{

printf("enter item which you want to search\n");

scanf("%d",&item);

ptr=header;

while(ptr->rlink!=NULL)

{

if(ptr->data==item)

{

flag=1;

loc=i+1;

break;

}

else

{

flag=0;

}

++i;

ptr=ptr->rlink;

}

if(flag==0)

{

printf("item not found\n");

}

else

{

printf("item found at location %d\n",loc);

}

}

}

struct node\*sort\_list(struct node\*header)

{

struct node\*ptr1,\*ptr2;

int temp;

ptr1=header;

while(ptr1->rlink!=NULL)

{

ptr2=ptr1->rlink;

while(ptr2!=NULL) //there are atleast 2 nodes in the list

{

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

{

temp=ptr1->data;

ptr1->data=ptr2->data;

ptr2->data=temp;

}

ptr2=ptr2->rlink;

}

ptr1=ptr1->rlink;

}

printf("list sorted\n");

return header;

}

