**Exercise 11: Write a program to implement following operations on the circular linked list. (a) Insert a node at the end of the linked list. (b) Insert a node before specified position. (c) Delete a first node of the linked list. (d) Delete a node after specified position.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*link;

};

struct node \*head = NULL, \*x, \*y, \*z;

void create();

void ins\_at\_beg();

void ins\_at\_pos();

void del\_at\_beg();

void del\_at\_pos();

void traverse();

void search();

void sort();

void update();

void rev\_traverse(struct node \*p);

void main()

{

int ch;

printf("\n 1.Creation \n 2.Insertion at beginning \n 3.Insertion at remaining");

printf("\n4.Deletion at beginning \n5.Deletion at remaining \n6.traverse");

printf("\n7.Search\n8.sort\n9.update\n10.Exit\n");

while (1)

{

printf("\n Enter your choice:");

scanf("%d", &ch);

switch(ch)

{

case 1:

create();

break;

case 2:

ins\_at\_beg();

break;

case 3:

ins\_at\_pos();

break;

case 4:

del\_at\_beg();

break;

case 5:

del\_at\_pos();

break;

case 6:

traverse();

break;

case 7:

search();

break;

case 8:

sort();

break;

case 9:

update();

break;

case 10:

rev\_traverse(head);

break;

default:

exit(0);

}

}

}

/\*Function to create a new circular linked list\*/

void create()

{

int c;

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

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

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

x->link = x;

head = x;

printf("\n If you wish to continue press 1 otherwise 0:");

scanf("%d", &c);

while (c != 0)

{

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

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

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

x->link = y;

y->link = head;

x = y;

printf("\n If you wish to continue press 1 otherwise 0:");

scanf("%d", &c);

}

}

/\*Function to insert an element at the begining of the list\*/

void ins\_at\_beg()

{

x = head;

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

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

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

while (x->link != head)

{

x = x->link;

}

x->link = y;

y->link = head;

head = y;

}

/\*Function to insert an element at any position the list\*/

void ins\_at\_pos()

{

struct node \*ptr;

int c = 1, pos, count = 1;

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

if (head == NULL)

{

printf("cannot enter an element at this place");

}

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

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

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

scanf("%d", &pos);

x = head;

ptr = head;

while (ptr->link != head)

{

count++;

ptr = ptr->link;

}

count++;

if (pos > count)

{

printf("OUT OF BOUND");

return;

}

while (c < pos)

{

z = x;

x = x->link;

c++;

}

y->link = x;

z->link = y;

}

/\*Function to delete an element at any begining of the list\*/

void del\_at\_beg()

{

if (head == NULL)

printf("\n List is empty");

else

{

x = head;

y = head;

while (x->link != head)

{

x = x->link;

}

head = y->link;

x->link = head;

free(y);

}

}

/\*Function to delete an element at any position the list\*/

void del\_at\_pos()

{

if (head == NULL)

printf("\n List is empty");

else

{

int c = 1, pos;

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

scanf("%d", &pos);

x = head;

while (c < pos)

{

y = x;

x = x->link;

c++;

}

y->link = x->link;

free(x);

}

}

/\*Function to display the elements in the list\*/

void traverse()

{

if (head == NULL)

printf("\n List is empty");

else

{

x = head;

while (x->link != head)

{

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

x = x->link;

}

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

}

}

/\*Function to search an element in the list\*/

void search()

{

int search\_val, count = 0, flag = 0;

printf("\nenter the element to search\n");

scanf("%d", &search\_val);

if (head == NULL)

printf("\nList is empty nothing to search");

else

{

x = head;

while (x->link != head)

{

if (x->data == search\_val)

{

printf("\nthe element is found at %d", count);

flag = 1;

break;

}

count++;

x = x->link;

}

if (x->data == search\_val)

{

printf("element found at postion %d", count);

}

if (flag == 0)

{

printf("\nelement not found");

}

}

}

/\*Function to sort the list in ascending order\*/

void sort()

{

struct node \*ptr, \*nxt;

int temp;

if (head == NULL)

{

printf("empty linkedlist");

}

else

{

ptr = head;

while (ptr->link != head)

{

nxt = ptr->link;

while (nxt != head)

{

if (nxt != head)

{

if (ptr->data > nxt->data)

{

temp = ptr->data;

ptr->data = nxt->data;

nxt->data = temp;

}

}

else

{

break;

}

nxt = nxt->link;

}

ptr = ptr->link;

}

}

}

/\*Function to update an element at any position the list\*/

void update()

{

struct node \*ptr;

int search\_val;

int replace\_val;

int flag = 0;

if (head == NULL)

{

printf("\n empty list");

}

else

{

printf("enter the value to be edited\n");

scanf("%d", &search\_val);

fflush(stdin);

printf("enter the value to be replace\n");

scanf("%d", &replace\_val);

ptr = head;

while (ptr->link != head)

{

if (ptr->data == search\_val)

{

ptr->data = replace\_val;

flag = 1;

break;

}

ptr = ptr->link;

}

if (ptr->data == search\_val)

{

ptr->data = replace\_val;

flag = 1;

}

if (flag == 1)

{

printf("\nUPdate sucessful");

}

else

{

printf("\n update not successful");

}

}

}

/\*Function to display the elements of the list in reverse order\*/

void rev\_traverse(struct node \*p)

{

int i = 0;

if (head == NULL)

{

printf("empty linked list");

}

else

{

if (p->link != head)

{

i = p->data;

rev\_traverse(p->link);

printf(" %d", i);

}

if (p->link == head)

{

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

}

}

}