//C program to Implement a Circular Linked List and develop functions to perform insertion, deletion and linear search operations.

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head;

// function prototyping

struct node \*create(int);

void insert\_at\_begin(int);

void insert\_at\_end(int);

void insert\_at\_position(int, int);

void delete\_at\_begin();

void delete\_at\_end();

void delete\_at\_position(int);

void search(int);

void update(int, int);

void print\_list();

void print\_list\_reverse();

int size\_of\_list();

int getData();

int getPosition();

int main()

{

char user\_active = 'Y';

int user\_choice;

int data, position;

while(user\_active == 'Y' || user\_active == 'y')

{

printf("\n\n------ Circular Singly Linked List -------\n");

printf("\n1. Insert a node at beginning");

printf("\n2. Insert a node at end");

printf("\n3. Insert a node at given position");

printf("\n\n4. Delete a node from beginning");

printf("\n5. Delete a node from end");

printf("\n6. Delete a node from given position");

printf("\n\n7. Print list from beginning");

printf("\n8. Print list from end");

printf("\n9. Search a node data");

printf("\n10. Update a node data");

printf("\n11. Exit");

printf("\n\n------------------------------\n");

printf("\nEnter your choice: ");

scanf("%d", &user\_choice);

printf("\n------------------------------\n");

switch(user\_choice)

{

case 1:

printf("\nInserting a node at beginning");

data = getData();

insert\_at\_begin(data);

break;

case 2:

printf("\nInserting a node at end");

data = getData();

insert\_at\_end(data);

break;

case 3:

printf("\nInserting a node at the given position");

data = getData();

position = getPosition();

insert\_at\_position(data, position);

break;

case 4:

printf("\nDeleting a node from beginning\n");

delete\_at\_begin();

break;

case 5:

printf("\nDeleting a node from end\n");

delete\_at\_end();

break;

case 6:

printf("\nDelete a node from given position\n");

position = getPosition();

delete\_at\_position(position);

break;

case 7:

printf("\nPrinting the list from beginning\n\n");

print\_list();

break;

case 8:

printf("\nPrinting the list from end\n\n");

if (head == NULL) {

printf("\n\tList is Empty!\n");

} else {

print\_list\_reverse(head);

}

break;

case 9:

printf("\nSearching the node data");

data = getData();

search(data);

break;

case 10:

printf("\nUpdating the node data");

data = getData();

position = getPosition();

update(position, data);

break;

case 11:

printf("\nProgram was terminated\n\n");

return 0;

default:

printf("\n\t Invalid Choice\n");

}

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

printf("\nDo you want to continue? (Y/N) : ");

fflush(stdin);

scanf(" %c", &user\_active);

}

return 0;

}

struct node \*create(int data)

{

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

if (new\_node == NULL)

{

printf("\nMemory can't be allocated.\n");

return NULL;

}

new\_node->data = data;

new\_node->next = NULL;

return new\_node;

}

// function to insert a new node at the beginning of the list

void insert\_at\_begin(int data)

{

struct node \*new\_node = create(data);

if (new\_node != NULL)

{

struct node \*last = head;

// if the list is empty

if (head == NULL)

{

head = new\_node;

new\_node->next = head;

return;

}

// traverse to the end node

while (last->next != head)

{

last = last->next;

}

// update the last node to the new node

last->next = new\_node;

// update the next pointer of the new node to the head node

new\_node->next = head;

// update the head of the list to new node

head = new\_node;

}

}

// function to insert a new node at the end of the list

void insert\_at\_end(int data)

{

struct node \*new\_node = create(data);

if (new\_node != NULL)

{

// if the list is empty

if (head == NULL)

{

head = new\_node;

new\_node->next = head;

return;

}

struct node \*last = head;

// traverse to the end node

while (last->next != head)

{

last = last->next;

}

// update the last node to the new node

last->next = new\_node;

// update the next pointer of the new node to the head node

new\_node->next = head;

}

}

// function to insert a new node at the given position

void insert\_at\_position(int position, int data)

{

// checking if the position is valid or not

if (position <= 0)

{

printf("\nInvalid Position\n");

}

else if (head == NULL && position > 1)

{

printf("\nInvalid Position\n");

}

else if (head != NULL && position > size\_of\_list())

{

printf("\nInvalid Position\n");

}

else if (position == 1)

{

insert\_at\_begin(data);

}

else

{

struct node \*new\_node = create(data);

if (new\_node != NULL)

{

struct node \*temp = head, \*prev = NULL;

// Since, temp is already pointing to first node

// then count will be start at second node

int i = 1;

// traverse the list to the given position

while (++i <= position)

{

prev = temp;

temp = temp->next;

}

// update the prev node to the new noe

prev->next = new\_node;

// update the new node to the temp (position node)

new\_node->next = temp;

}

}

}

// function to delete a node from the beginning of the list

void delete\_at\_begin()

{

// check where the list is empty or not

if (head == NULL)

{

printf("\n List is Empty! \n");

return;

}

// traverse to the end of the list

struct node \*last = head;

struct node \*temp = head;

// if only one node in the list

if (last->next == head)

{

free(last);

head = NULL;

return;

}

// traverse to the last node

while (last->next != head)

{

last = last->next;

}

head = head->next;

last->next = head;

free(temp);

temp = NULL;

}

// function to delete a node from the end of the list

void delete\_at\_end()

{

// check where the list is empty or not

if (head == NULL)

{

printf("\n List is Empty! \n");

return;

}

// traverse to the end of the list

struct node \*prev = head;

struct node \*temp = head->next;

// if only one node in the list

if (prev->next == head)

{

free(prev);

head = NULL;

return;

}

while (temp->next != head)

{

prev = temp;

temp = temp->next;

}

prev->next = head;

free(temp);

temp = NULL;

}

// function to delete a node from the given position

void delete\_at\_position(int position)

{

if (position <= 0)

{

printf("\n Invalid Position \n");

}

else if (position > size\_of\_list())

{

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

}

else if (position == 1)

{

delete\_at\_begin();

}

else if (position == size\_of\_list())

{

delete\_at\_end();

}

else

{

struct node \*temp = head;

struct node \*prev = NULL;

int i = 1;

while (i < position)

{

prev = temp;

temp = temp->next;

i += 1;

}

prev->next = temp->next;

free(temp);

temp = NULL;

}

}

// print the node values

void print\_list()

{

struct node \*temp = head;

if (head == NULL)

{

printf("\n List is Empty! \n");

return;

}

printf("\n");

do

{

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

temp = temp->next;

} while (temp != head);

printf("\n");

}

// print the node values recursively

void print\_list\_reverse(struct node\* temp)

{

if (temp->next == head)

{

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

return;

}

print\_list\_reverse(temp->next);

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

}

// search a data into the list

void search(int key)

{

struct node\* temp = head;

do

{

if (temp->data == key)

{

printf("\n\t Data Found\n");

return;

}

temp = temp->next;

}while (temp->next != head);

printf("\n\tData not Found\n");

}

// function to update a node

void update(int position, int new\_data)

{

if (position <= 0 || position > size\_of\_list())

{

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

return;

}

struct node\* temp = head;

int i = 0;

while (i <= position)

{

temp = temp->next;

i += 1;

}

temp->data = new\_data;

}

// function to calculate the size of the list

int size\_of\_list()

{

if (head == NULL)

{

return 0;

}

struct node \*temp = head;

int count = 1;

while (temp->next != head)

{

count += 1;

temp = temp->next;

}

return count;

}

int getData()

{

int data;

printf("\n\nEnter Data: ");

scanf("%d", &data);

return data;

}

int getPosition()

{

int pos;

printf("\n\nEnter Position: ");

scanf("%d", &pos);

return pos;

}