Circular LinledList

Circular linkedlist is a regular linkedlist where the last element of the list points to the head of the list.

Algorithms

let getnode be a function that initializes a new node and returns its pointer

Creation

1. Start
2. newprt = getnode()
3. if (nodeptr = NULL)
   nodeptr = newptr
   else
   last->next = newptr
   newptr->next = nodeptr
4. last = newprt
5. repeat from 2 if user wants to add more
6. stop

Traversing

1. Start
2. ptr = nodeptr
3. do
   print ptr->info
   ptr = ptr->next
   while (ptr != nodeptr)
4. stop

Insertion

1. Start
2. newptr = getnode()
3. ptr = nodeptr
4. do
   ptr = ptr->next
   while(ptr->info != key AND ptr != nodeptr)
5. if (ptr->info != key)
   print "key not found"
   else
   newptr->next = ptr->next
   ptr->next = newptr
6. Stop

Deletion

1. Start
2. if(nodeptr = NULL)
   error "LinkedList is empty"
   else
   i. ptr = nodeptr
   ii. do
   prevptr = ptr
   ptr = ptr->next
   while(ptr->info != key AND ptr != nodeptr)
   iii. if(ptr->info != key)
   print "Node with key doesnt exist"
   else
   prevptr->next = ptr->next
   free(ptr)
3. Stop

IMPLEMENTATION

```
#include <stdio.h>
#include <stdlib.h>


struct Node {
    int info;
    struct Node* next;
};


struct Node* getnode() {
    struct Node* newptr = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter data for the new node: ");
    scanf("%d", &(newptr->info));
    return newptr;
}


struct Node* nodeptr = NULL;
struct Node* last = NULL;

void createList() {
    char choice;

    do {
        struct Node* newptr = getnode();


        if (nodeptr == NULL) {
```

```
            nodeptr = newptr;
        } else {
            last->next = newptr;
            newptr->next = nodeptr;
        }

        last = newptr;

        printf("Do you want to add more nodes? (y/n): ");
        scanf(" %c", &choice);
    } while (choice == 'y' || choice == 'Y');
}


void traverseList() {
    if (nodeptr == NULL) {
        printf("Circular LinkedList is empty.\n");
        return;
    }

    struct Node* ptr = nodeptr;

    do {
        printf("%d ", ptr->info);
        ptr = ptr->next;
    } while (ptr != nodeptr);

    printf("\n");
}

void insertNode(int key) {
    if (nodeptr == NULL) {
        printf("Circular LinkedList is empty. Cannot insert.\n");
        return;
    }

    struct Node* newptr = getnode();

    struct Node* ptr = nodeptr;

    do {
        ptr = ptr->next;
    } while (ptr->info != key && ptr != nodeptr);

    if (ptr->info != key) {
        printf("Node with key %d not found.\n", key);
    } else {
        newptr->next = ptr->next;
        ptr->next = newptr;
    }
}
```

```
void deleteNode(int key) {
    if (nodeptr == NULL) {
        printf("Circular LinkedList is empty. Cannot delete.\n");
        return;
    }

    struct Node *ptr = nodeptr, *prevptr;

    do {
        prevptr = ptr;
        ptr = ptr->next;
    } while (ptr->info != key && ptr != nodeptr);

    if (ptr->info != key) {
        printf("Node with key %d not found.\n", key);
    } else {
        prevptr->next = ptr->next;
        if(ptr == nodeptr)
            nodeptr = ptr->next;
        free(ptr);
    }
}

int main() {
    int ch, x;
    printf("CIRCULAR LINKEDLIST IMPLEMENTATION.\n\n");
    createList();
    while(1) {
        printf("1. Insert.\n"
            "2. Remove.\n"
            "3. Print\n"
            "4. Exit\n"
            ">> ");

        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("Enter the key of node after which new node should be
inserted: ");
            scanf("%d", &x);
            insertNode(x);
            break;
        case 2:
            printf("Enter the key of node which is to be removed: ");
            scanf("%d", &x);
            deleteNode(x);
            break;
        case 3:
            traverseList();
            break;
        case 4:
            exit(0);
```

```
            break;
        default:
            printf("Invalid option %d\n", ch);
            break;
        }
    }

    return 0;
}
```

OUTPUT

```
CIRCULAR LINKEDLIST IMPLEMENTATION.

Enter data for the new node: 10
Do you want to add more nodes? (y/n): y
Enter data for the new node: 20
Do you want to add more nodes? (y/n): n
1. Insert.
2. Remove.
3. Print
4. Exit
>> 3
10 20
1. Insert.
2. Remove.
3. Print
4. Exit
>> 1
Enter the key of node after which new node should be inserted: 10
Enter data for the new node: 15
1. Insert.
2. Remove.
3. Print
4. Exit
>> 3
10 15 20
1. Insert.
2. Remove.
3. Print
4. Exit
>> 2
Enter the key of node which is to be removed: 10
1. Insert.
2. Remove.
3. Print
4. Exit
>> 3
15 20
1. Insert.
```

```
2. Remove.
3. Print
4. Exit
>> 1
Enter the key of node after which new node should be inserted: 30
Enter data for the new node: 10
Node with key 30 not found.
1. Insert.
2. Remove.
3. Print
4. Exit
>>
```