**Assignment 6 | 9th January 2021 DSA**

**Question 1**

**Write a program implementing insert, delete and display operation of Circular Queue.**

#include<stdio.h>

#include<stdlib.h>

struct Node;

typedef struct Node \* PtrToNode;

typedef PtrToNode List;

typedef PtrToNode Position;

struct Node

{

int e;

Position next;

};

void Insert(int x, List l, Position p)

{

Position TmpCell;

TmpCell = (struct Node\*) malloc(sizeof(struct Node));

if(TmpCell == NULL)

printf("Memory out of space\n");

else

{

TmpCell->e = x;

TmpCell->next = p->next;

p->next = TmpCell;

}

}

int isLast(Position p, List l)

{

return (p->next == l);

}

Position FindPrevious(int x, List l)

{

Position p = l;

while(p->next != l && p->next->e != x)

p = p->next;

return p;

}

Position Find(int x, List l)

{

Position p = l->next;

while(p != l && p->e != x)

p = p->next;

return p;

}

void Delete(int x, List l)

{

Position p, TmpCell;

p = FindPrevious(x, l);

if(!isLast(p, l))

{

TmpCell = p->next;

p->next = TmpCell->next;

free(TmpCell);

}

else

printf("Element does not exist!!!\n");

}

void Display(List l)

{

printf("The list element are :: ");

Position p = l->next;

while(p != l)

{

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

p = p->next;

}

}

int main()

{

int x, pos, ch, i;

List l, l1;

l = (struct Node \*) malloc(sizeof(struct Node));

l->next = l;

List p = l;

printf("CIRCULAR LINKED LIST IMPLEMENTATION OF LIST ADT\n\n");

do

{

printf("\n\n1. INSERT\t 2. DELETE\t 3. FIND\t 4. PRINT\t 5. QUIT\n\nEnter the choice :: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

p = l;

printf("Enter the element to be inserted :: ");

scanf("%d",&x);

printf("Enter the position of the element :: ");

scanf("%d",&pos);

for(i = 1; i < pos; i++)

{

p = p->next;

}

Insert(x,l,p);

break;

case 2:

p = l;

printf("Enter the element to be deleted :: ");

scanf("%d",&x);

Delete(x,p);

break;

case 3:

Display(l);

break;

}

}while(ch<4);

return 0;

}

**Question 3**

**Implement push, pop and find the minimum element in a stack in O(1) time complexity.**

#include <iostream>

#include <stack>

class Stack

{

// main stack to store elements

std::stack<int> s;

// variable to store minimum element

int min;

public:

// Inserts a given element on top of the stack

void push(int x)

{

if (s.empty()) {

s.push(x);

min = x;

}

else if (x > min) {

s.push(x);

}

else {

s.push(2 \* x - min);

min = x;

}

}

// Removes top element from the stack and returns it

void pop()

{

if (s.empty()) {

std::cout << "Stack underflow!!" << '\n';

}

int top = s.top();

if (top < min)

min = 2 \* min - top;

s.pop();

}

// Returns the minimum element from the stack in constant time

int minimum()

{

return min;

}

};

int main()

{

Stack s;

s.push(6);

std::cout << s.minimum() << '\n';

s.push(7);

std::cout << s.minimum() << '\n';

s.push(5);

std::cout << s.minimum() << '\n';

s.push(3);

std::cout << s.minimum() << '\n';

s.pop();

std::cout << s.minimum() << '\n';

s.pop();

std::cout << s.minimum() << '\n';

return 0;

}