**Double Ended Queue**

Problem Statement:

Write a program to implement a double ended queue using doubly linked list. It should be a menu driven program with

following options:

1. Insert a new node at head

2. Insert a new node at end

3. Insert after kth node

4. Delete from first

5. Delete from end

6. Delete the kth node

7. Print the list

8. Search an item in the list

9.exit

• Output example :

Type 1 to insert at head

Type 2 to insert at tail

Type 3 to insert at any pos

Type 4 to delete the first element

Type 5 to delete at last position

Type 6 to delete at a given position

Type 7 to print the list

Type 8 to search

Type 9 to terminate

1

Enter value: 15

2

Enter value: 14

3

Enter value & position: 13 1

7

15 ->13 ->14

6

Enter position: 1

2

Enter value: 13

7

15 ->14 ->13

1

Enter value: 16

1

Enter value: 17

7

17 ->16 ->15 ->14 ->13

4

7

16 ->15 ->14 ->13

5

7

16 ->15 ->14

2

Enter value: 13

2

Enter value: 12

7

16 ->15 ->14 ->13 ->12

6

Enter position: 3

7

16 ->15 ->14 ->12

8

Enter value: 14

2

9

Terminated

Proposed C Code:

/\* ------- main.c ------- \*/

#include <stdio.h>

#include <stdlib.h>

typedef struct Node

{

int data;

struct Node \*next;

struct Node \*prev;

} Queue;

Queue \*head;

Queue \*tail;

int size = 0;

Queue \*newNode(int x) //Creating a new element

{

Queue \*newNode = (Queue \*)malloc(sizeof(Queue));

newNode->data = x;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

void insertatfirst(int val) // Insert at first position

{

Queue \*temp = newNode(val);

if (head == NULL)

{

tail = head = temp;

}

else

{

temp->next = head;

head->prev = temp;

head = temp;

}

size++;

}

void insertatlast(int val) // Insert at last position

{

Queue \*temp = newNode(val);

if (tail == NULL)

{

tail = head = temp;

}

else

{

temp->prev = tail;

tail->next = temp;

tail = temp;

}

size++;

}

void insertatpos(int val, int pos) // Insert at given position

{

if (pos > size)

{

printf("Size Exceed\n");

return;

}

if (pos == size)

{

insertatlast(val);

}

if (pos == 0)

{

insertatfirst(val);

}

Queue \*temp = newNode(val);

int i = pos - 1;

Queue \*p = head;

while (i > 0)

{

p = p->next;

i--;

}

Queue \*new = p->next;

p->next = temp;

temp->next = new;

size++;

}

int search(int val) // Find he index of a node

{

Queue \*temp = head;

int i = 0;

while (temp != NULL)

{

if (temp->data == val)

{

return i;

}

i++;

temp = temp->next;

}

return -1;

}

void deleteatfirst() // Delete the first node

{

if (head == NULL)

{

printf("Queue is Empty\n");

}

else

{

Queue \*temp = head;

head = head->next;

if (head == NULL)

tail = NULL;

else

head->prev = NULL;

free(temp);

}

size--;

}

void deleteatlast() // Delete the last node

{

if (tail == NULL)

{

printf("Queue is Empty\n");

}

else

{

Queue \*temp = tail;

tail = tail->prev;

if (tail == NULL)

head = NULL;

else

tail->next = NULL;

free(temp);

}

size--;

}

void deleteatpos(int pos) // Delete the node of the given position

{

if (pos > size)

{

printf("Size Exceed\n");

return;

}

if (pos == size)

{

deleteatlast();

}

if (pos == 0)

{

deleteatfirst(head);

}

Queue \*temp = head;

int i = pos - 1;

while (i > 0)

{

temp = temp->next;

i--;

}

temp->next = temp->next->next;

size--;

}

void display() // Display the entire Queue

{

Queue \*temp = head;

while (temp != tail)

{

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

temp = temp->next;

}

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

printf("\n");

}

int main()

{

head = NULL;

tail = NULL;

printf("Type 1 to insert at head\nType 2 to insert at tail \nType 3 to insert at any pos \nType 4 to delete the first element \nType 5 to delete at last position \nType 6 to delete at a given position\nType 7 to print the list\nType 8 to search \nTYpe 9 to terminate\n");

int x, pos, val;

do

{

scanf("%d", &x);

switch (x)

{

case 1:

printf("Enter value: ");

scanf("%d", &val);

insertatfirst(val);

break;

case 2:

printf("Enter value: ");

scanf("%d", &val);

insertatlast(val);

break;

case 3:

printf("Enter value & position: ");

scanf("%d %d", &val, &pos);

insertatpos(val, pos);

break;

case 4:

deleteatfirst();

break;

case 5:

deleteatlast();

break;

case 6:

printf("Enter position: ");

scanf("%d", &pos);

deleteatpos(pos);

break;

case 7:

display();

break;

case 8:

printf("Enter value: ");

scanf("%d", &val);

printf("%d\n", search(val));

break;

case 9:

break;

}

} while (x != 9);

printf("Terminated");

return 0;

}

/\* ---------------------- \*/

Conclusion:

The proposed algorithm has a runtime of O(1).

Limitations and assumptions for this algorithm include:

1.Here we are using two pointers instead of one and another two dummy pointer.

2.Any input except 1 to 9 is invalid.