Navlakhi®

Doubly Ended Queues

Methodology and Program

By Abhishek Navlakhi
Semester 3: Data Structures

Program

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


struct node
{
int data;
struct node *link;
};


struct queue
{
int count;
struct node *front;
struct node *rear;
}*q;


void createQueue( )
{
q=(struct queue *)malloc(sizeof(struct queue));
if (q ==NULL)
{
printf("Insufficient memory.....\n");
exit(1);
}
q ->front= NULL;
q ->rear=NULL;
q ->count=0;
}
```

```
int addend(int dataIn)
{
struct node *pNew;

pNew=(struct node *)malloc(sizeof(struct node));

if (pNew!=NULL)
{
    pNew->data=dataIn;
    pNew->link=NULL;

    if (q->count==0)
          q->front=pNew;
    else
          q->rear->link=pNew;

    q->rear=pNew;
    q->count+=1;
    return 1;
}
else
    return 0;
}
```

```
int delfront(int *dataPtr)
{
struct node *pLoc;

if (q->count==0) return 0;

pLoc=q->front;
*dataPtr=pLoc->data; /****OR *dataPtr=q->front->data   ***/

if (q->count==1) q->rear=NULL;

q->front=q->front->link; /*OR q->front=pLoc->link */
q->count-=1;

free(pLoc);
return 1;
}
```

```
int addfront(int data)
{
  struct node *pNew;
  pNew=(struct node*)malloc(sizeof(struct node));
  if (pNew!=NULL)
  {
    pNew->data=data;
    if(q->count==0)
          q->front=q->rear=pNew;
    else
    {
      pNew->link=q->front;
      q->front=pNew;
    }
    q->count++;
    return 1;
  }
  else
  {
    printf("no memory");
    return 0;
  }
}
```

```
int delend(int *data)
{
  struct node *pLoc,*pPrev;
  if (q->count==0)
 {
    printf("Queue is empty");
    return 0;
 }

  pLoc=q->rear;
  *data=pLoc->data;
  if(q->count==1)
  {
    q->front=q->rear=NULL;
  }
  else
  {
    pPrev=q->front;
    while(pPrev->link!=pLoc)
          pPrev=pPrev->link;

    q->rear=pPrev;
    pPrev->link=NULL;
  }
  free(pLoc);
  q->count--;
  return 1;
}
```

```
int queueFront(int *dataPtr)
{
if (q->count==0) return 0;

*dataPtr=q->front->data;
return 1;
}


int queueRear(int *dataPtr)
{
if (q->count==0) return 0;

*dataPtr=q->rear->data;
return 1;
}


void destroyQueue( )
{
struct node *pLoc;
while (q->count!=0)
{
pLoc=q->front;
q->front=q->front->link; /* OR q->front=pLoc->link */
q->count-=1;
free(pLoc);
}

free(q);
}
```

```
int fullQueue( )
{
struct node *temp;
temp=(struct node*)malloc(sizeof(struct node*));
if (temp==NULL)    return 1;
else { free(temp);  return 0; }
}


int menu( )
{
int choice;
printf("\n\n\t\t M E N U \n");
printf("\n\n\t\t ========\n");
printf("1. Add Data To end of Queue\n");
printf("2. Remove Data From start of Queue\n");
printf("3. add front\n");
printf("4. delete end\n");
printf("5. View Front element\n");
printf("6. View Rear element\n");
printf("7. View Count\n");
printf("8. Check if Memory FULL\n");
printf("9. exit\n");

printf("\n Feed in your choice: ");
scanf("%d",&choice);

return choice;
}
```

```
void main( )
{
int choice,dataIn,dataOut,success;

createQueue();

do
{
choice=menu();
switch(choice)
{
case 1:  printf("feed in Data to insert: ");
    scanf("%d",&dataIn);
    success=addend(dataIn);
    if(success) printf("Data inserted successfully\n");
    else printf("data Insertion Failed.. Insufficient memory..\n");
    break;

case 2:  success=delfront(&dataOut);
    if (success) printf("Data successfully Deleted = %d\n",dataOut);
    else printf("Data Not present\n");
    break;

case 3:  printf("feed in Data to insert: ");
    scanf("%d",&dataIn);
    success=addfront(dataIn);
    if(success) printf("Data inserted successfully\n");
    else printf("data Insertion Failed.. Insufficient memory..\n");
    break;

case 4:  success=delend(&dataOut);
    if (success) printf("Data successfully Deleted = %d\n",dataOut);
    else printf("Data Not present\n");
    break;
```

```
case 5: success=queueFront(&dataOut);
    if (success) printf("Data at the front of queue= %d\n",dataOut);
    else printf("No data in queue\n");
    break;

case 6: success=queueRear(&dataOut);
    if (success) printf("Data at the rear of queue= %d\n",dataOut);
    else printf("No data in queue\n");
    break;

case 7: printf("Number of Data Items in Queue = %d\n",q->count);
    break;

case 8:  success=fullQueue();
    if (success) printf("Memory FULL !!!! ");
    else printf("Memory not FULL...\n");
    break;
}

}while(choice!=9);

destroyQueue();
}
```

HOME OF EDUCATION

Navlakhi®

www.navlakhi.com
Home of Education

DATA Structures@ Navlakhi's