CIS 2101 [Data Structures and Algorithms]

Date: Sept. 16, 2020

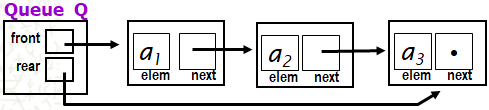
Assignment #5 : **Pairwise Activity on Queue**

**For each of the following implementations of Queue, write an appropriate definition of data type Queue, code of the functions:**

1. **initQueue() – initialize the queue to be empty.**
2. **enqueue() – inserts an element at the rear of the queue.**
3. **dequeue() – deletes the front element of the queue.**
4. **front() – returns the front element of the queue.**

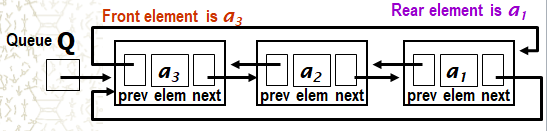
**Note: Check the functions in a program before submitting. Only functions are required in this activity.**

1. Singly-linked list Implementation of ADT Queue



|  |  |
| --- | --- |
| **Definition of the Queue**  **typedef struct cell{**  **char elem;**  **struct cell \* next;**  **}\*celltype;** | **typedef struct{**  **celltype front;**  **celltype rear;**  **}Queue;** |
| **initQueue()**  **void initQueue(Queue \*Q)**  **{**  **Q->front = NULL;**  **Q->rear= NULL;**  **}** | **front()**  **char front(Queue Q)**  **{**  **char ret = '0';**    **if(Q.front!=NULL){**  **ret = Q.front->elem;**  **}**  **return ret;**  **}** |
| **enqueue()**  **void enqueue(Queue \*Q, char ins)**  **{**  **celltype temp;**    **temp=(celltype)malloc(sizeof(struct cell));**  **if(temp!=NULL){**  **temp->elem = ins;**  **temp->next = NULL;**  **if(Q->rear==NULL){**  **Q->front = temp;**  **}else{**  **(Q->rear)->next = temp;**  **}**  **Q->rear = temp;**  **}**  **}** | **dequeue()**  **void dequeue(Queue \*Q)**  **{**  **celltype temp;**    **if(Q->front!=NULL){**  **temp = Q->front;**  **Q->front = (Q->front)->next;**  **if(Q->front==NULL){**  **Q->rear=NULL;**  **}**  **free(temp);**  **}**  **}** |

1. Doubly-linked list Implementation of ADT Queue



|  |  |
| --- | --- |
| **Definition of the Queue**  **typedef struct cell{**  **struct cell \* prev;**  **char elem;**  **struct cell \* next;**  **}\*Queue;** |  |
| **initQueue()**  **void initQueue(Queue \*Q)**  **{**  **\*Q=NULL;**  **}** | **front()**  **char front(Queue Q)**  **{**  **char ret = '0';**    **if(Q!=NULL){**  **ret = Q->elem;**  **}**  **return ret;**  **}** |
| **enqueue()**  **void enqueue(Queue \*Q, char ins)**  **{**  **Queue temp;**    **temp=(Queue)malloc(sizeof(struct cell));**  **if(temp!=NULL){**  **temp->elem = ins;**  **if(\*Q==NULL){**  **\*Q = temp;**  **temp->next = \*Q;**  **temp->prev = \*Q;**  **}else{**  **temp->next = \*Q;**  **temp->prev = (\*Q)->prev;**  **(temp->prev)->next = temp;**  **(\*Q)->prev = temp;**  **}**  **}**  **}** | **dequeue()**  **void dequeue(Queue \*Q)**  **{**  **Queue temp;**    **if(\*Q!=NULL){**  **if((\*Q)!=(\*Q)->next){**  **((\*Q)->prev)->next=(\*Q)->next;**  **((\*Q)->next)->prev=(\*Q)->prev;**  **temp=\*Q;**  **\*Q=(\*Q)->next;**  **free(temp);**  **}else{**  **temp=\*Q;**  **\*Q=NULL;**  **free(temp);**  **}**  **}**  **}** |

1. Circular Array Implementation of ADT Queue. Queue is considered full if there are Max – 1 elements where Max is the size of the array. Queue is manipulated in a clockwise operation.

|  |  |
| --- | --- |
|  | **Definition of the Queue**  **#define Max 8**  **typedef struct cell{**  **char Elem[Max];**  **int front;**  **int rear;**  **}Queue;** |
| **initQueue()**  **void initQueue(Queue \*Q)**  **{**  **Q->front = 5;**  **Q->rear = 4;**  **}** | **front()**  **char front(Queue Q)**  **{**  **char ret = '0';**    **if(Q.front != (Q.rear+1)%Max){**  **ret = Q.Elem[Q.front];**  **}**  **return ret;**  **}** |
| **enqueue()**  **void enqueue(Queue \*Q, char ins)**  **{**  **if(Q->front != (Q->rear+2)%Max){**  **Q->rear = (Q->rear+1)%Max;**  **Q->Elem[Q->rear] = ins;**  **}**  **}** | **dequeue()**  **void dequeue(Queue \*Q)**  **{**  **if(Q->front != (Q->rear+1)%Max){**  **Q->front = (Q->front+1)%Max;**  **}**  **}** |

1. **Submission Details:**
   1. Filename: Queue\_Lastnames .
   2. Email to [chetpena@gmail.com](mailto:chetpena@gmail.com) with **email subject**: CIS 2101 – Ass #5
   3. PLEASE follow instructions.