UCSC Silicon Valley Extension

Advanced C Programming

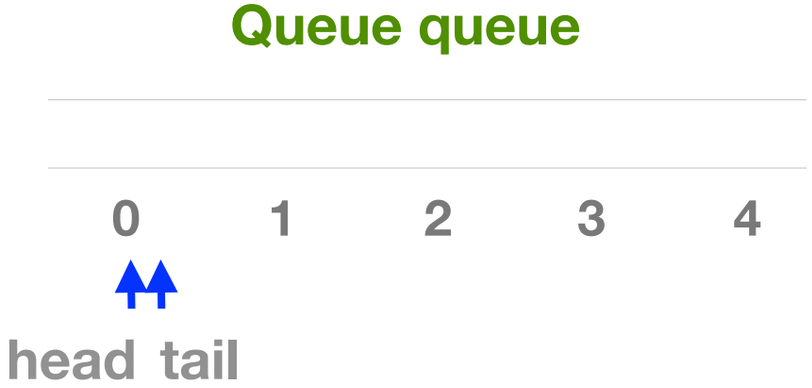
Queues

Radhika Grover

Overview

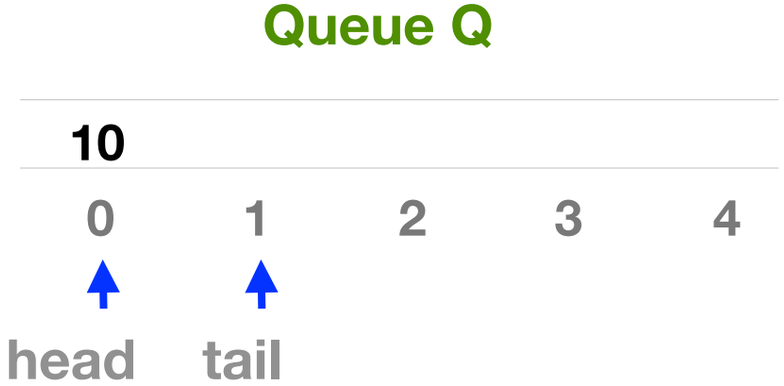
- Queues
- Example
- Implementation using an array
- Applications

Circular queue example



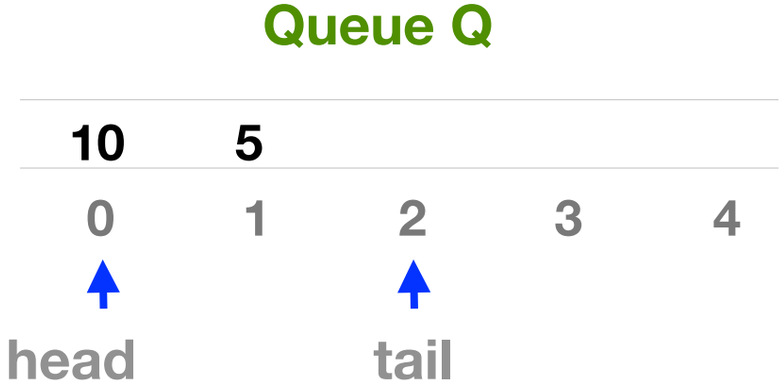
enQueue(queue, 10)

Circular queue example



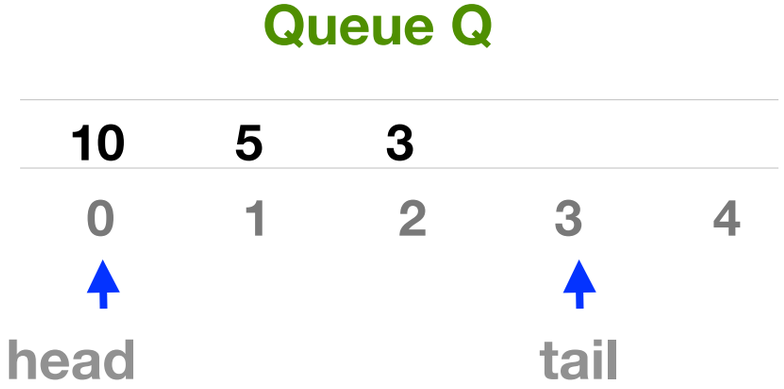
enQueue(queue, 5)

Circular queue example



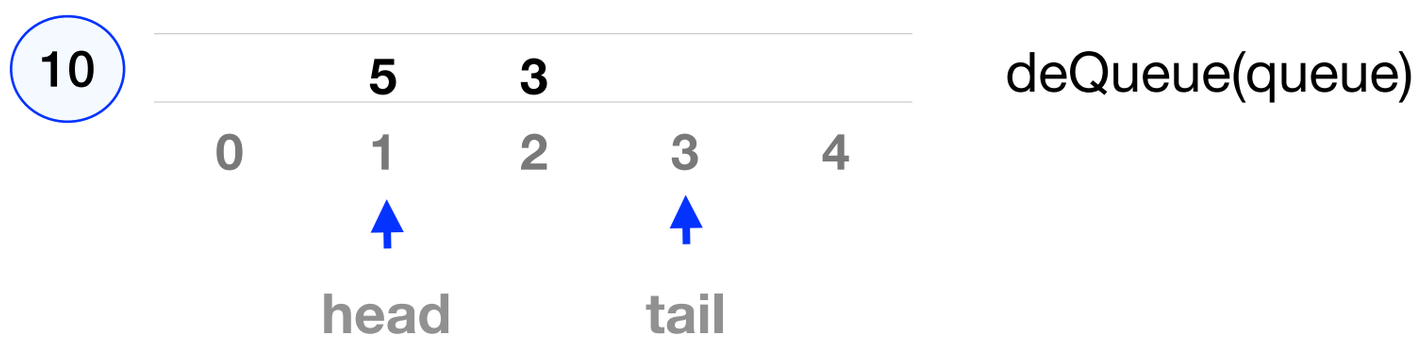
enQueue(queue, 3)

Circular queue example

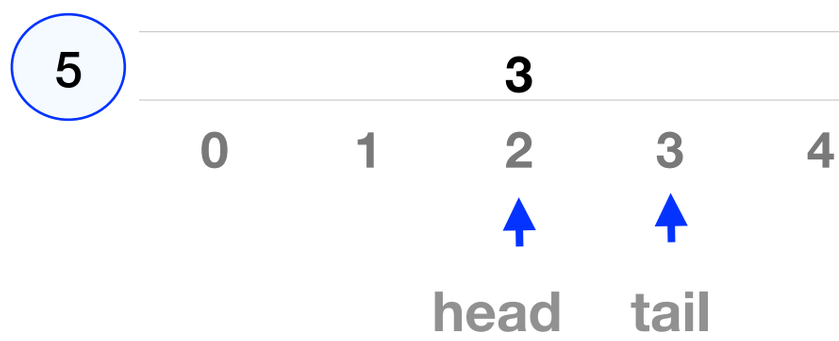


deQueue(queue)

Circular queue example



Circular queue example



Circular queue example

enQueue(queue, 1)

Circular queue example

enQueue(queue, 1)

Circular queue example

Array based implementation pseudocode:

```
enqueue(int x){
    if (size == capacity)
       throwError();
   else {
            array[tail] = x;
             ++size;
            tail = (tail + 1) %
            capacity;
        }
}
```

```
dequeue(){
    if (size == 0)
        throwError();
    int value = array[head];
   --size;
    head = (head + 1) % capacity;

    return value;
}
```

Time complexity = O(1)

Circular queue using a linked list

```c
// CircularQueue/circularQueue.h

// define Data according to requirements
typedef struct Data_t {
    int id;
    int distance;
    int predecessor;
} Data;

// a node in the queue
typedef struct Node_t {
    Data value;
    struct Node_t *next;
} Node;

// queue has a head that points to first node and a tail that points to the last node.
typedef struct Queue_t {
    Node *head;
    Node *tail;
    unsigned int size;
} Queue;
```

Circular queue using a linked list

```
// CircularQueue/circularQueue.h
// create an empty queue
Queue* createQueue() {
    Queue *queue = (Queue *)  calloc(sizeof(Queue), 1);
    if (queue == NULL) {
        fprintf(stderr, "Error: Queue could not be created");
        exit(EXIT_FAILURE);
    }
    *queue = (Queue) {.head = NULL, .tail = NULL, .size = 0};
    return queue;
}
```

Circular queue using a linked list

```
//CircularQueue/circularQueue.c

// create a node with specified Data and add it to the end of the queue
void enqueue(Queue **queue, Data newValue) {
    Node *newNode = (Node *) calloc(sizeof(Node), 1);

    if (newNode == NULL) {
        printf("%s", "Error: memory could not be allocated");
        exit (EXIT_FAILURE);
    } else {
        newNode->value = newValue;
        newNode->next = NULL;

        if ((*queue)->head == NULL) {
            (*queue)->head = newNode;
            (*queue)->tail = newNode;
            (*queue)->tail->next = NULL;
        }
        else {
            (*queue)->tail->next = newNode;
            (*queue)->tail = newNode;
            (*queue)->tail->next = NULL;
        }

        ++(*queue)->size;
    }
}
```

Circular queue using a linked list

```
// remove and return the node at the front of the queue
Node* dequeue(Queue **queue){
    if ((*queue)->head == NULL) {
        printf("%s","empty queue");
        return NULL;
    }
    else {
        Node *tmp = (*queue)->head;
        (*queue)->head = (*queue)->head->next;

        --(*queue)->size;
        return tmp;
    }

}

// return true if queue is empty
bool isEmpty(Queue *queue){
    if (queue->size == 0)
        return true;
    else
        return false;
}
```

Circular queue using a linked list

```
int main(void) {

    Queue *queue = createQueue();
    Data data1 = {10, 100, 20};
    enqueue(&queue, data1);

    Data data2 = {20, 200, 40};
    enqueue(&queue, data2);

    Data data3 = {30, 300, 60};
    enqueue(&queue, data3);

    print(queue);
// … remaining code
}
```

Check for memory leaks

- Check your program for memory leaks.

- For example, if you are using valgrind, run it on the command line for the executable Stack as:

- valgrind --tool=memcheck --leak-check=full --show-leak-kinds=all ./ Stack