Computer Science >

Data Structures: Queues (with C Program source code)

Quick Links to Various Sections on the Main Page

The Learning Point
Computer Science
Algorithms: An introduction to Dynamic Programming
CS - Data Structures and Algorithms
CS - Programming Interviews
CS - Miscellaneous C Programs
CS - Intro to DB Queries
Electrical Science and Engineering
Electrical Sci - DC Circuits
Electrical Sci - Digital Electronics
Mathematics
Mathematics - Linear Algebra
Mathematics - Geometry
Mathematics - Single Variable Calculus
Mathematics - Game

To go through the C program / source-code, scroll down to the end of this page

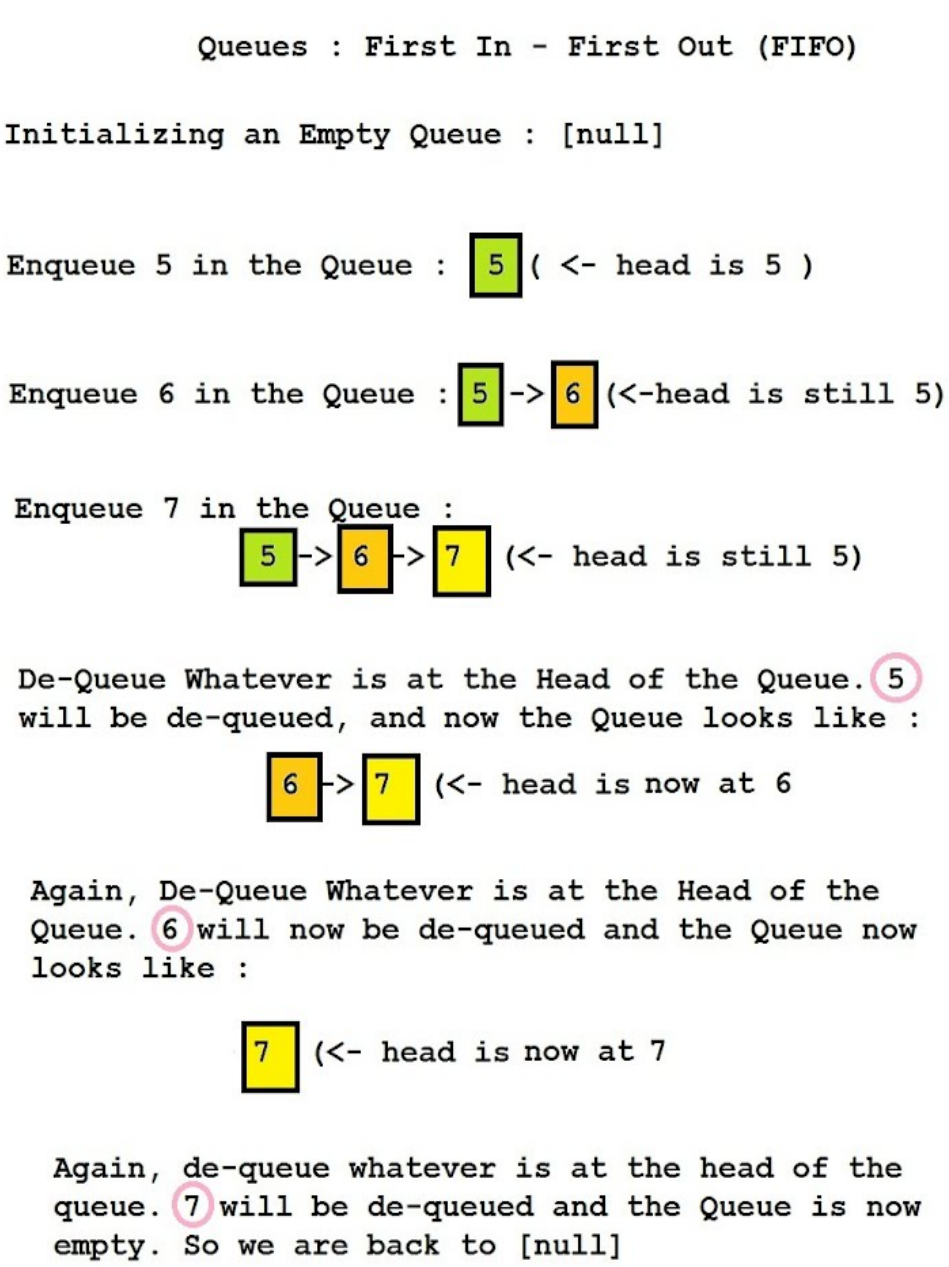
Queue

Queue is a specialized data storage structure (Abstract data type). Unlike, arrays access of elements in a Queue is restricted. It has two main operations enqueue and dequeue. Insertion in a queue is done using enqueue function and removal from a queue is done using dequeue function. An item can be inserted at the end ('rear') of the queue and removed from the front ('front') of the queue. It is therefore, also called First-In-First-Out (FIFO) list. Queue has five properties - capacity stands for the maximum number of elements Queue can hold, size stands for the current size of the Queue, elements is the array of elements, front is the index of first element (the index at which we remove the element) and rear is the index of last element (the index at which we insert the element).

Theory
Mathematics - Operations Research
Physics
Physics - Basic Mechanics
Physics - Electrostatics
Sitemap
Recent site activity

These are Quick Links to sections of the Main Page

1



Algorithm:

Queue structure is defined with fields capacity, size, *elements
(pointer to the array of elements), front and rear.

Functions –

1. createQueue function– This function takes the maximum number
of elements
(maxElements) the Queue can hold as an argument, creates a Queue
according to it
and returns a pointer to the Queue.
2. enqueue function - This function takes the pointer to the top of

the queue Q and the item
(element) to be inserted as arguments. Check for the emptiness of queue
3. dequeue function - This function takes the pointer to the top of the stack S as an
argument and will then dequeue an element.
4. front function – This function takes the pointer to the top of the queue Q as an argument and returns the front element of the queue Q.

Properties:

1. Each function runs in O(1) time.
2. It has two basic implementations
Array-based implementation – It's simple and efficient but the maximum size of
the queue is fixed.
Singly Linked List-based implementation – It's complicated but there is no limit
on the queue size, it is subjected to the available memory.

Complete Tutorial with document :

Queue

Queue is a specialized data storage structure (Abstract data type). Unlike, arrays access of elements in a Queue is restricted. It has two main operations **enqueue** and **dequeue**. Insertion in a queue is done using **enqueue** function and removal from a queue is done using **dequeue** function. An item can be inserted at the end ('rear') of the queue and removed from the front ('front') of the queue. It is therefore, also called First-In-First-Out (FIFO) list. Queue has five properties - capacity stands for the maximum number of elements Queue can hold, size stands for the current size of the Queue, elements is the array of elements, front is the index of first element (the index at which we remove the element) and rear is the index of last element (the index at which we insert the element).

Algorithm:

Queue structure is defined with fields capacity, size, *elements (pointer to the array of elements), front and rear.

Functions –

1. createQueue function– This function takes the maximum number of elements (maxElements) the Queue can hold as an argument, creates a Queue according to it and returns a pointer to the Queue. It initializes Queue Q using malloc function and its properties.
   elements = (int *)malloc(sizeof(int)*maxElements).
   Q->size = 0, current size of the Queue Q.
   Q->capacity = maxElements, maximum number of elements Queue Q can hold.
   Q->front = 0

Queues - C Program source code

```
#include<stdio.h>
#include<stdlib.h>
/*Queue has five properties. capacity stands for the maximum number of elements Queue can hold.
  Size stands for the current size of the Queue and elements is the array of elements. front is the
 index of first element (the index at which we remove the element) and rear is the index of last element
 (the index at which we insert the element) */
typedef struct Queue
{
        int capacity;
        int size;
        int front;
        int rear;
        int *elements;
}Queue;
/* crateQueue function takes argument the maximum number of elements the Queue can hold, creates
   a Queue according to it and returns a pointer to the Queue. */
Queue * createQueue(int maxElements)
{
        /* Create a Queue */
        Queue *Q;
        Q = (Queue *)malloc(sizeof(Queue));
        /* Initialise its properties */
        Q->elements = (int *)malloc(sizeof(int)*maxElements);
        Q->size = 0;
        Q->capacity = maxElements;
        Q->front = 0;
        Q->rear = -1;
        /* Return the pointer */
        return Q;
}
void Dequeue(Queue *Q)
{
        /* If Queue size is zero then it is empty. So we cannot pop */
        if(Q->size==0)
        {
                printf("Queue is Empty\n");
                return;
        }
        /* Removing an element is equivalent to incrementing index of front by one */
        else
        {
                Q->size--;
                Q->front++;
                /* As we fill elements in circular fashion */
                if(Q->front==Q->capacity)
                {
                        Q->front=0;
                }
        }
        return;
}
int front(Queue *Q)
{
        if(Q->size==0)
        {
                printf("Queue is Empty\n");
                exit(0);
```

```
        }
        /* Return the element which is at the front*/
        return Q->elements[Q->front];
}
void Enqueue(Queue *Q,int element)
{
        /* If the Queue is full, we cannot push an element into it as there is no space for it.*/
        if(Q->size == Q->capacity)
        {
                printf("Queue is Full\n");
        }
        else
        {
                Q->size++;
                Q->rear = Q->rear + 1;
                /* As we fill the queue in circular fashion */
                if(Q->rear == Q->capacity)
                {
                        Q->rear = 0;
                }
                /* Insert the element in its rear side */
                Q->elements[Q->rear] = element;
        }
        return;
}
int main()
{
        Queue *Q = createQueue(5);
        Enqueue(Q,1);
        Enqueue(Q,2);
        Enqueue(Q,3);
        Enqueue(Q,4);
        printf("Front element is %d\n",front(Q));
        Enqueue(Q,5);
        Dequeue(Q);
        Enqueue(Q,6);
        printf("Front element is %d\n",front(Q));
}
```

Related Tutorials :

Stacks	Last In First Out data structures (LIFO). Like a stack of cards from which you pick up the one on the top (which is the last one to be placed on top of the stack). Documentation of the various operations and the stages a stack passes through when elements are inserted or deleted. C program to help you get an idea of how a stack is implemented in code.	Queues	First in First Out data structure (FIFO). Like people waiting to buy tickets in a queue - the first one to stand in the queue, gets the ticket first and gets to leave the queue first. Documentation of the various operations and the stages a queue passes through as elements are inserted or deleted. C Program source code to help you get an idea of how a queue is implemented in code.

Some Important Data Structures and Algorithms, at a glance:

Arrays : Popular Sorting and Searching Algorithms			
Bubble Sort	Insertion Sort	Selection Sort	Shell Sort

Merge Sort	Quick Sort	Heap Sort	Binary Search Algorithm
Basic Data Structures and Operations on them			
Stacks	Queues	Single Linked List	Double Linked List
Circular Linked List	1.		

Tree Data Structures			
Binary Search Trees	Heaps	Height Balanced Trees	
Graphs and Graph Algorithms			
Depth First Search	Breadth First Search	Minimum Spanning Trees: Kruskal Algorithm	Minumum Spanning Trees: Prim's Algorithm
Dijkstra Algorithm for Shortest Paths	Floyd Warshall Algorithm for Shortest Paths	Bellman Ford Algorithm	
Popular Algorithms in Dynamic Programming			
Dynamic Programming	Integer Knapsack problem	Matrix Chain Multiplication	Longest Common Subsequence
Greedy Algorithms			
Elementary cases : Fractional Knapsack Problem, Task Scheduling	Data Compression using Huffman Trees		