

# Абстрактный тип данных очередь

---

Примеры  
использования  
абстрактной  
очереди

# Абстрактный тип данных

## Очередь

---

- *Очередью* называется последовательность элементов одного и того же типа, к которой можно добавлять новые элементы и из которой можно удалять элементы.

Причем добавление элементов производится в один конец , а удаление происходит с другого конца последовательности.

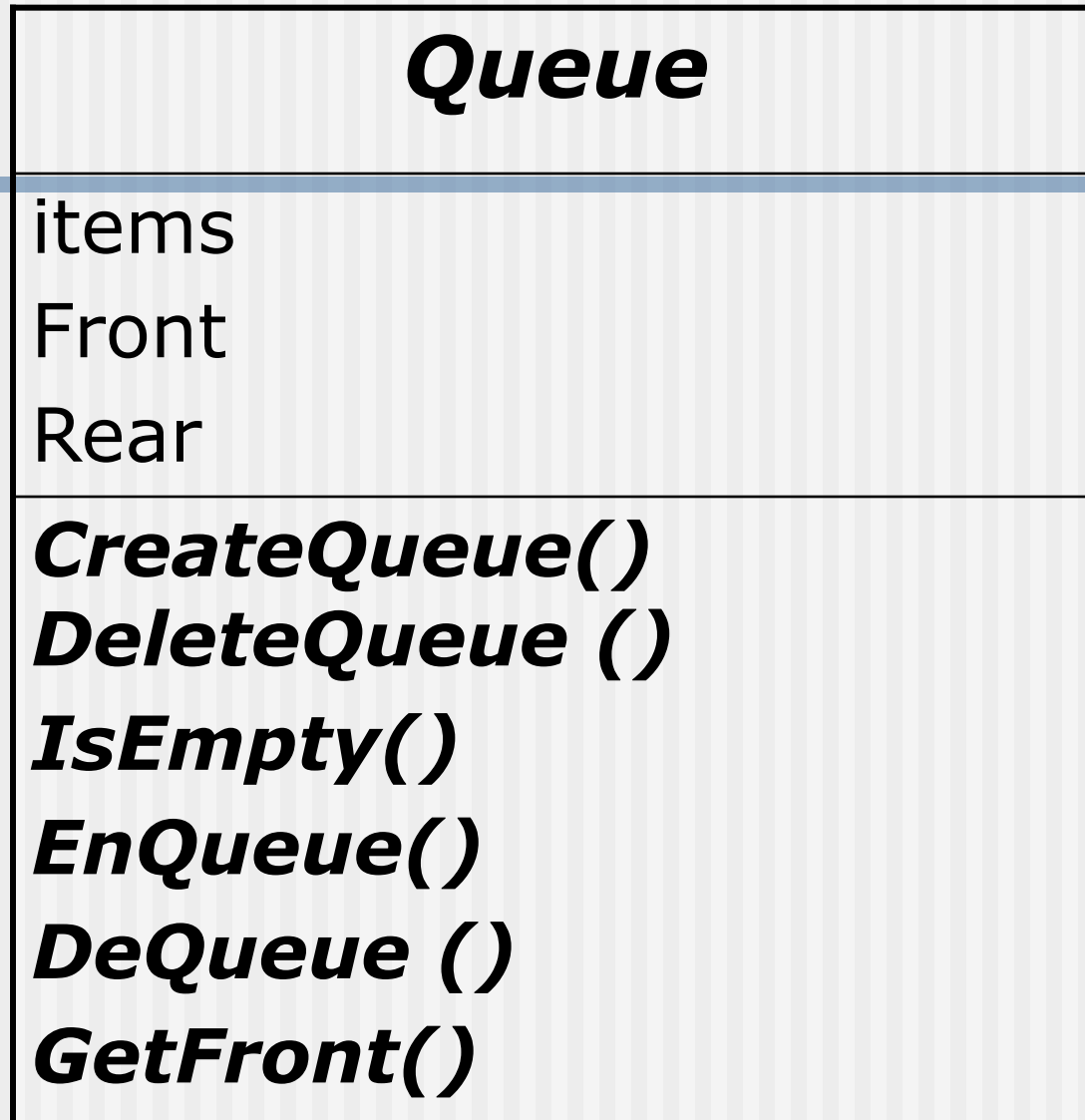
# Абстрактный тип данных

## Очередь

---

- Конец очереди, из которого выполняется удаление элементов, называется началом или *головой очереди* (Front)
- Конец очереди, куда происходит вставка элементов, называется *хвостом очереди* (Rear)

# Диаграмма абстрактной очереди



# Операции с очередью

---

- ***CreateQueue()*** - создает пустую очередь
- ***DeleteQueue ()*** – уничтожает очередь
- ***IsEmpty()*** – определяет пуста ли очередь
- ***EnQueue(NewElement)*** – добавляет новый элемент *NewElement* в конец очередь
- ***DeQueue ()*** – удаляет элемент из начала очереди
- ***GetFront()*** – возвращает значение элемента из начала без его удаления

# Очередь

```
graph TD; A[Очередь] --> B[Ограниченная]; A --> C[Неограниченная];
```

Ограниченная	Неограниченная
Количество элементов ограничивается некоторым числом	Размер ограничен только доступной памятью
Реализуется с помощью кольцевого массива	Реализуется в виде связного списка, связного кольцевого списка или в виде абстрактного списка

# Реализация ограниченной очереди в виде массива

---

- Размер массива определяет максимальное число элементов в очереди
- Необходимо определить указатель `front` положения первого элемента и указатель `rear` положения последнего элемента

# Реализация ограниченной очереди в виде массива

---

Пусть `TypeItem` – тип элементов стека

`Max_queue` – максимальный размер очереди

```
Struct Queue {  
    TypeItem Items [Max_queue]; //массив  
    элементов очереди  
    int front; //индекс первого элемента  
    int rear; //индекс последнего элемента  
    int count; //кол-во элементов  
}
```



# Реализация ограниченной очереди в виде массива

---

- При вставке и удалении элементов индексы `front` (при удалении) и `rear` (при вставке) перемещаются вперед вдоль массива по часовой стрелке
- Условие опустошенной или полной очереди: индекс `front` непосредственно предшествует индексу `rear`
- Для определения полноты очереди необходимо ввести дополнительный счетчик `count`
  - `Count` увеличивается на единицу при добавлении элемента
  - `Count` уменьшается на единицу при удалении элемента

# Реализация ограниченной очереди в виде массива

---

```
Void CreateQueue(Queue Q)
```

```
{ Q.front=0;
```

```
  Q.rear=-1;
```

```
}
```

```
Void EnQueue(Queue Q,TypeItem NewItem)
```

```
{ if (Q.count==Max_Queue)
```

```
  cout>>'Очередь полна';
```

```
  else
```

```
    Q.rear=(Q.rear+1)% Max_Queue;
```

```
    Q.Item[Q.rear]=NewItem;
```

```
    ++Q.count;
```

```
} //end EnQueue
```

# Основные операции с очередью

```
Void DeQueue(Queue Q)  
{ if ( IsEmpty(Q))  
    cin>>'Очередь пуста';  
  else  
{ Q.front=(Q.front+1)%Max_Queue;  
  --Q.count;  
  }//end if  
}//end DeQueue
```

```
TypeItem GetFront(Queue Q)  
{if (Q.count==0)    cin>>'  
очередь пуста';  
  else  
    return(Q.Items[Q.front];  
}// end GetFront
```

```
Int IsEmpty(Queue Q)  
{ return(Q.count==0)  
}// end IsEmpty
```

# Реализация очереди в виде СВЯЗНОГО СПИСКА

---

Пусть TypeItem – тип элементов стека

```
Struct QueueNode      // Узел очереди
```

```
{
```

```
    TypeItem Item;
```

```
    QueueNode * next;
```

```
};
```

```
Struct Queue
```

```
{
```

```
    QueueNode *front; // Указатель на первый элемент
```

```
    QueueNode *rear; // Указатель на последний элемент
```

```
}
```