Tạo queue từ danh sách liên kết đơn

- ✓ Tổng quan về queue
- ✓ Các thao tác trên queue
- ✓ Ví dụ minh họa

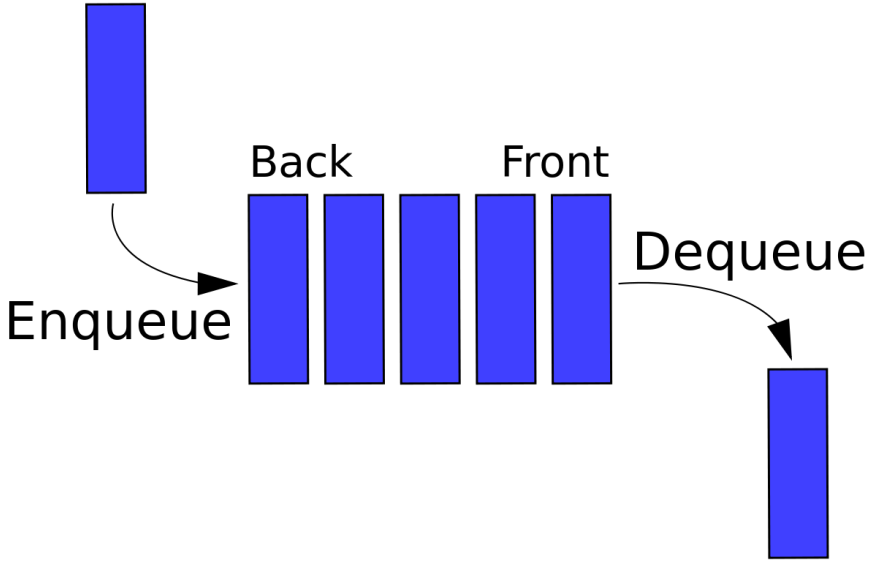
Tổng quan về queue

- ✓ Queue là một cấu trúc dữ liệu thường triển khai theo quy tắc FIFO-first in first out: vào trước ra trước.
- ✓ Queue luôn mở ở hai đầu, trong đó một đầu luôn dùng để chèn dữ liệu vào và một đầu còn lại luôn dùng để lấy dữ liệu ra.
- ✓ Quy tắc quản lý dữ liệu trong queue là phần tử nào được đưa vào trước sẽ được lấy ra trước.
- ✓ Queue có thể được triển khai bằng cách sử dụng mảng, danh sách liên kết,…
- ✓ Ví dụ về queue: dòng phương tiện nối đuôi nhau trên đường 1 làn xe; dòng người nối đuôi nhau làm thủ tục checkin ở sân bay…

Ứng dụng của queue

- ✓ Queue được sử dụng trong các ứng dụng cần quản lý một nhóm các đối tượng mà phần tử nào tới trước sẽ được xử lý trước.
- ✓ Sử dụng để phục vụ yêu cầu đối với các tài nguyên dùng chung: máy in, CPU, bộ nhớ.
- ✓ Sử dụng khi dữ liệu được truyền một cách đồng bộ giữa hai tiến trình.
- ✓ Trung tâm hỗ trợ khách hàng, tổng đài sử dụng queue để giữ các cuộc gọi đến theo thứ tự cho tới khi cuộc gọi đó được phục vụ.
- ✓ Trong các hệ thống xử lý ngắt thời gian thực, các ngắt được xử lý theo cùng thứ tự mà nó xuất hiện.
- ✓ Sử dụng trong hàng đợi mail, chuyển mạch router, switches

Minh họa về queue



Các thao tác trên queue

- ✓ **Push()** – thêm phần tử mới vào cuối queue.
- ✓ **Pop()** – xóa phần tử ở đầu queue.
- ✓ **Peek()** – lấy phần tử đầu queue nhưng không xóa.
- ✓ **Back()** – lấy phần tử cuối của queue.
- ✓ **IsEmpty()** – kiểm tra xem queue có rỗng không.
- ✓ **Elements()** – liệt kê các phần tử trong queue.

Tạo node

```
// lớp mô tả thông tin một node
6 references
class Node<T>
{
    3 references
    public T Data { get; set; } // phần dữ liệu của lớp
    5 references
    public Node<T> Next { get; set; } // phần liên kết

    1 reference
    public Node()
    {
        Next = null;
    }

    1 reference
    public Node(T data) : this()
    {
        Data = data;
    }
}
```

Tạo queue

```
3 references
class Queue<T>
{
    private Node<T> First;
    private Node<T> Last;

    1 reference
    public Queue()
    {
        First = null;
        Last = null;
    }
    // thêm node vào queue
    6 references
    public void Push(T data)...
    // xóa node đầu queue
    3 references
    public void Pop()...
    // lấy phần tử đầu queue
    3 references
    public T Peek()...
    // lấy phần tử cuối
    0 references
    public T Back()...
    // kiểm tra rỗng
    8 references
    public bool IsEmpty()...
    // liệt kê các phần tử trong queue
    1 reference
    public void Elements()...
}
```

Thao tác push

```
// thêm node vào queue
6 references
public void Push(T data)
{
    var node = new Node<T>(data);
    if (IsEmpty())
    {
        node.Next = First;
        First = node;
        Last = node;
    } else
    {
        Last.Next = node;
        Last = node;
    }
}
```

Thao tác pop, peek

```
// xóa node đầu queue
3 references
public void Pop()
{
    if (!IsEmpty())
    {
        var top = First;
        First = First.Next;
        top.Next = null;
    }
    else
    {
        throw new Exception("Queue Is Empty.");
    }
}
// lấy phần tử đầu queue
3 references
public T Peek()
{
    if (!IsEmpty())
    {
        return First.Data;
    }
    else
    {
        throw new Exception("Queue Is Empty.");
    }
}
```

Thao tác back, isempty

```
// lấy phần tử cuối
0 references
public T Back()
{
    if(!IsEmpty())
    {
        return Last.Data;
    }
    throw new Exception("Queue Is Empty.");
}

// kiểm tra rỗng
8 references
public bool IsEmpty()
{
    return First == null;
}
```

Thao tác elements

```
// liệt kê các phần tử trong queue
1 reference
public void Elements()
{
    if (!IsEmpty())
    {
        var x = First;
        while (x != null)
        {
            Console.Write($"{x.Data} -> ");
            x = x.Next;
        }
        Console.WriteLine("null");
    }
    else
    {
        Console.WriteLine("==> Queue rỗng.");
    }
}
```

Nội dung tiếp theo

Tìm hiểu các cấu trúc dữ liệu generic