1. 数组实现一个栈

public class Stack {

**//存数据的数组**

int[] data;

**//栈的最大长度**

private int size;

**//栈顶的位置**

private int top;

**//栈的构造函数**

public Stack(int size)

{

this.size = size;

data = new int[size];

top = -1;

}

**/\*\* \* 判断是否为空栈 \* @return \*/**

public boolean isEmpty()

{

return top == -1;

}

**/\*\* \* 判断是否为满栈 \* @return \*/**

public boolean isFull()

{

return (top+1) == size;

}

**/\*\* \* 压栈操作 \* @param data \* @return \*/**

public boolean push(int data)

{

if(isFull())

{

System.out.println("the stack is full!");

return false;

} else

{

top++;

this.data[top] = data;

return true; }

}

**/\*\* \* 弹栈操作 \* @return \* @throws Exception \*/**

public int pop() throws Exception

{

if(isEmpty())

{

throw new Exception("the stack is empty!");

} else

{ return this.data[top--];

}

}

**/\*\* \* 获取栈顶的元素,但不弹栈 \* @return \*/**

public int peek()

{

return this.data[getTop()];

}

1. 数组实现一个队列 泛型

public class ArrayQueue<E> implements Queue<E> {

    private E[] data;

    private int size ;//队列中对象数量

    private int front ;//队列中第一个对象位置

private int rear ;//队列中当前对象位置

    public ArrayQueue() {

        data = (E[]) new Object[10];

        size = 0;

        front = 0;

        rear = 0;

    }

**//尾部入队列**

    public void add(E target) {

        if( isFull() ){

            enlarge();

            front = 0;

        }

**rear = (front + size) % data.length;**

        data[rear] = target;

        size++;

    }

    public E front() {

        if (isEmpty()) {

            throw new RuntimeException("队列为空!");

        }

        return data[front];

    }

**//判断为空**

    public boolean isEmpty() {

        return size == 0;

    }

**//头部出队列**

    public E remove() {

        if (isEmpty()) {

            throw new RuntimeException("队列为空!");

        }

        E tempData = data[front];

        data[front] = null;

**front = (front + 1) % (data.length);**

        size--;

        return tempData;

    }

    public int size() {

        return size;

    }

**/\*\***

**\* 判断当前队列是否已满**

**\***

**\* @return**

**\*/**

    public boolean isFull() {

        return size == data.length;

    }

**/\*\***

**\* 将数组容量扩大两倍**

**\***

**\*/**

    public void enlarge() {

        E[] newData = (E[]) new Object[data.length \* 2];

        System.arraycopy(data, 0, newData, 0, data.length);

        data = newData;

        newData = null;

    }

}

快排，堆排序，非递归中序遍历，找环入口

单例模式。快排，链表逆序，判断链表环和找环入口、多线程卖票

快排，单链表逆转，单例模式，字符串转整型，判断两个单链表是否相交、归并，字符串去重

字典排序，不过面试官让我说下思路

字符串最大字典序的子序列（还要求优化到O(n)），strcpy，快排，合并有序单链表，判断一个数是不是质数

一个数组中连续子向量的最大和

手写二分查找

1. **反转单链表**

package javatest1;

public class JavaTest1 {

public static void main(String[] args) {

Node head = new Node(0);

Node node1 = new Node(1);

Node node2 = new Node(2);

Node node3 = new Node(3);

head.setNext(node1);

node1.setNext(node2);

node2.setNext(node3);

**// 打印反转前的链表**

**Node h = head;**

**while (null != h) {**

**System.out.print(h.getData() + " ");**

**h = h.getNext();**

**}**

// 调用反转方法

head = reverse2(head);

System.out.println("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// 打印反转后的结果

while (null != head) {

System.out.print(head.getData() + " ");

head = head.getNext();

}

}

/\*\*

\* 遍历，将当前节点的下一个节点缓存后更改当前节点指针

\*/

public static Node reverse2(Node head) {

**if (head == null)**

**return head;**

Node pre = head;// 上一结点

Node cur = head.getNext();// 当前结点

**Node tmp;**// 临时结点，用于保存当前结点的指针域（即下一结点）

while (cur != null) {// 当前结点为null，说明位于尾结点

**tmp = cur.getNext();**

**cur.setNext(pre);// 反转指针域的指向**

**// 指针往下移动**

**pre = cur;**

**cur = tmp;**

}

// 最后将原链表的头节点的指针域置为null，还回新链表的头结点，即原链表的尾结点

head.setNext(null);

return pre;

}

}

class Node {

private int Data;// 数据域

private Node Next;// 指针域

public Node(int Data) {

// super();

this.Data = Data;

}

public int getData() {

return Data;

}

public void setData(int Data) {

this.Data = Data;

}

public Node getNext() {

return Next;

}

public void setNext(Node Next) {

this.Next = Next;

}

}