TLV[1]编码是按Tag Length Value的格式进行编码的，一段码流中的信息只用Tag标识，Tag在码流中唯一不重复，Length表示信息Value的长度，Value表示信息数据。

码流以某个预先定义的Tag开头，Tag固定占一个字节，Length固定占两个字节，字节按小端序，为小端序。

现给定TLV格式编码的码流，以及需要解码的目标Tag，请输出该目标Value。

输入码流的16进制字符中，不包括小写字母，且要求输出的16进制字符串中也不要包含小写字母；码流字符串的最大长度不超过50000个字节。

输入描述

- 输入的第一行为一个字符串，表示待解码的目标Tag；
- 输入的第二行为一个字符串，表示待解码的16进制码流，字节之间有空格分隔。

输出描述

- 输出一个字符串，表示待解码信息以16进制表示的Value。

用例

输入	31 32 01 00 AE 90 02 00 01 02 30 03 0E AB 32 31 31 02 00 32 33 33 01 08 CC
输出	32 33
说明	需要解析的目标Tag是31， 从码流的第0位开始匹配， 第一个信息的Tag是32，但长长度为01（01 00，小端序表示为1）； 第二个信息的Tag是90，其长度为2； 第三个信息的Tag是30，其长度为3； 第四个信息的Tag是31，其长度为2（02 00）； 所以返回长度后面的两个字节即可，即32 33。

题目解析

本题最容易理解的应该是内存管理，但最大难度其实如下：

第二行输入是码流，是由多个信息组成的，每个信息又是由tag、len、val组成。

其中tag占一个字节，len占两个字节，而val占的字节数由len决定，因此上面给的第二行输入如下图：

tag	len		val		tag	len		val		tag	len		val		tag	len		val		tag	len		val	
32	01	00	AE		90	02	00	01	02	30	03	0E	AB	32	31	31	02	00	32	33	33	01	08	CC

可能这样大家就一目了然了吧。

现在要找tag为31的信息val，从上图可以看出val为32 33

JavaScript算法源码

```
// 注意 cin 输入输出模式需要调用 rl.close()
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    // 输入完毕，开始处理
    const tag = lines[0];

    // 输入的第二行是字符串
    const stream = lines[1].split(" ");

    console.log(getResult(stream, tag));

    lines.length = 0;
  }
});

function getResult(stream, find) {
  // 注意题目要求输入广义表是倒入内存栈中，所以需要用pop操作，pop的结果要求一致
  stream.reverse();

  while (stream.length) {
    const tag = stream.pop();

    const len = parseInt(
      // 注意字符串的长度，因此需要反转
      [stream.pop(), stream.pop(), "0x"].reverse().join(""),
      16
    );

    const val = [];
    for (let i = 0; i < len; i++) {
      val.push(stream.pop());
    }

    if (tag === find) {
      return val.join(" ");
    }
  }
}
```

Java算法源码

```
import java.util.ArrayList;
import java.util.Scanner;
import java.util.StringJoiner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    String target = sc.nextLine();
    String[] stream = sc.nextLine().split(" ");

    System.out.println(getResult(stream, target));
  }

  public static String getResult(String[] stream, String target) {
    int i = 0;

    while (i < stream.length) {
      String tag = stream[i+i];

      String tmp1 = stream[i+i];
      String tmp2 = stream[i+i];

      int len = Integer.parseInt(tmp2 + tmp1, 16);

      ArrayList<String> val = new ArrayList<>();
      for (int j = 0; j < len; j++) {
        val.add(stream[i+i]);
      }

      if (tag.equals(target)) {
        StringJoiner sj = new StringJoiner(" ");
        for (String s : val) sj.add(s);
        return sj.toString();
      }
    }

    return null;
  }
}
```

Python算法源码

```
# 输入处理
target = input()
stream = input().split()


# 算法入口
def getResult():
    stream.reverse()

    while len(stream) > 0:
        # 注意字符串的长度，因此需要反转的内存栈，所以需要用pop操作，pop的结果要求一致
        tag = stream.pop()

        tmp = [stream.pop(), stream.pop(), "0x"]
        # 注意字符串的长度，因此需要反转
        tmp.reverse()

        long = int("".join(tmp), 16)

        val = []
        for i in range(long):
            val.append(stream.pop())

        if tag == target:
            return " ".join(val)


# 算法调用
print(getResult())
```