### **全排列**

## 实现满足下面功能的汇编程序

1. 使用mips实现全排列生成算法。
2. 以0x00000000为数据段起始地址。
3. 输入一个小于等于7的正整数，求出n的全排列，并按照字典序输出。
4. 每组数据最多执行500,000条指令。

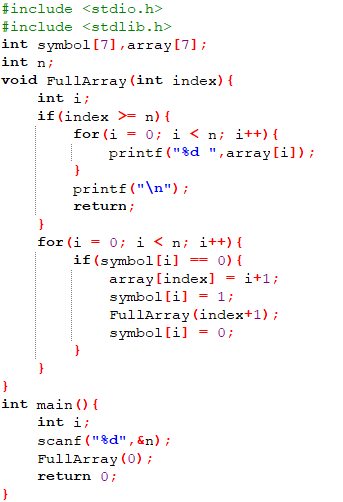
## 输入格式

只输入一行，输入一个整数n**(0<n<=7)**

## 输出格式

按照字典序输出n!行数组，每行输出n个数字，数字之间以空格隔开，每行最后一个数字后可以有空格。

## C代码提示



## 输入样例

4

## 输出样例

1 2 3 4

1 2 4 3

1 3 2 4

1 3 4 2

1 4 2 3

1 4 3 2

2 1 3 4

2 1 4 3

2 3 1 4

2 3 4 1

2 4 1 3

2 4 3 1

3 1 2 4

3 1 4 2

3 2 1 4

3 2 4 1

3 4 1 2

3 4 2 1

4 1 2 3

4 1 3 2

4 2 1 3

4 2 3 1

4 3 1 2

4 3 2 1

## 提交要求

1. **请勿使用**.globl main
2. 不考虑延迟槽
3. 只需要提交.asm文件。
4. 程序的初始地址设置为**Compact,Data at Address 0**。

### **回文串判断**

## 实现满足下面功能的汇编程序

1. 判断输入的字符串是不是回文串。
2. 输出一个字符，是回文串输出1，否则输出0
3. 每组数据最多执行100,000条指令。

## 输入格式

第一行为一个整数n，代表字符串的长度。第二行开始的n行：每行一个字符(小写字母)，连起来为输入的字符串。**(0<n<=20)**

## 输出格式

输出为一行，输出一个字符，是会文串输出1，否则输出0。

## 输入样例

5

a

b

b

d

l

## 输出样例

0

## 提交要求

1. **请勿使用**.globl main
2. 不考虑延迟槽
3. 只需要提交.asm文件。
4. 程序的初始地址设置为**Compact,Data at Address 0**。

## 注意事项

注意!因为评测机的行为和MARS有一些区别，你需要注意以下事项。

1. 如果你采取每次读入一个字符的系统调用（$v0=12）来读入数据,那么我们保证你不会读入到任何换行符。如果你采取这种方式输入，那么对于样例，你可以在MARS中首先手动输入5，打回车，然后手动在一行之中输入abbdl。
2. 如果你采取一次读一行的系统调用（$v0=8），那么你读入的每行有一个小写字母以及行尾的一个换行符。
3. 如果你的程序长时间等不到应有的输入，则有可能提示超时或运行错误。在你处理字符的时候，你需要考虑到上述情况。

### **矩阵乘法**

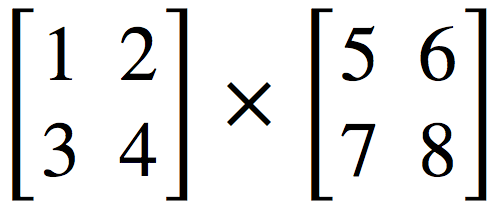
## **任务**

使用MIPS汇编语言编写一个具有矩阵相乘功能的汇编程序(不考虑延迟槽)。

## **具体要求**

* 首先读取方形矩阵的阶数n，然后再依次读取第一个矩阵（n行n列）和第二个矩阵（n行n列）中的元素。
* 两个矩阵的阶数相同，我们提供的测试数据中0<n≤8,每个矩阵元素小于10。
* 最终将计算出的结果输出，每行n个数据，每个数据间用空格分开。评测机会自动过滤掉行尾空格以及最后的回车。

## **样例**



比如我们想要计算上面这两个矩阵相乘的结果，我们会给出以下输入：

2

1

2

3

4

5

6

7

8

正确的输出应该是：

19 22

43 50

## **提交要求**

* 请勿使用.globl main。
* 不考虑延迟槽。
* 只需要提交.asm文件。
* 程序的初始地址设置（Mars->Settings->Memory Configuration）为**Compact,Data at Address 0**。

### **选择排序**

## 实现满足下面功能的MIPS汇编程序

1. 给定n个整数(1 ≤ n ≤ 100)从小到大排序。
2. n行每行一个整数(数据范围[1,100000])，为排序前的数列
3. 输出一行,为从小到大排序后的数列,以空格分割。
4. 每组数据最多执行100,000条指令。

## 输入格式

第一行是一个整数，代表上文中的n。接下来n行每行一个整数(数据范围[1,100000])，为排序前的数列。

## 输出格式

输出一行,为从小到大排序后的数列,以空格分割。

## 输入样例

5

2

1

3

3

2

## 输出样例

1 2 2 3 3

## 约定

1. n为整数且1 ≤ n ≤ 100
2. 对于输出，评测机会过滤掉最后的换行，和每行最后的空格、制表符。

## 提交要求

1. **请勿使用**.globl main
2. 不考虑延迟槽
3. 只需要提交.asm文件。
4. 程序的初始地址设置为**Compact,Data at Address 0**。

### **内存操作**

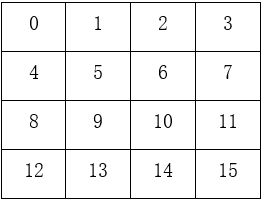
## **简介**

在高级语言中，我们可以使用多维数组对内存进行多维操作，但实际上，一般这些多维数组在内存中也只是按照一维的形式连续存储的。比如二维数组arr[2][2]，他在内存中会占用4个单位的连续空间，分别保存arr[0][0]、arr[0][1]、arr[1][0]、arr[1][1]。

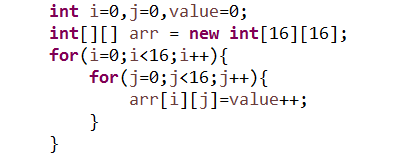
作为低级语言，汇编语言对内存只能进行一维操作。而为了实现多维操作，我们就需要使用一些技巧了，下面会进行举例分析。

## **样例分析**

比如我们要将0~255依次赋给一个16\*16的矩阵，填充方式如下图4\*4矩阵的例子。

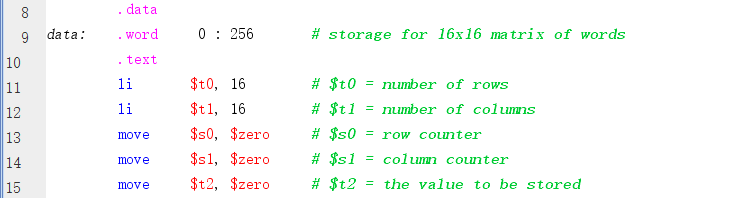


 使用高级语言的话只需要两个for循环嵌套，如下：



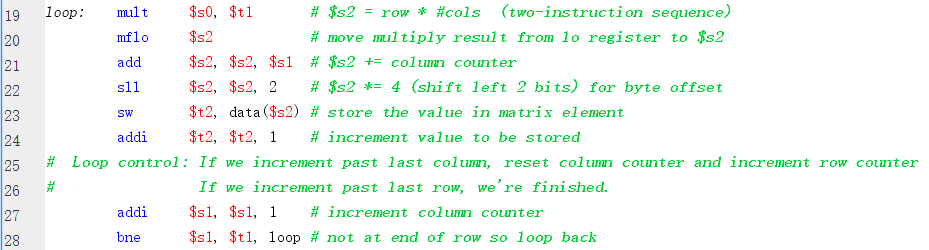
使用汇编语言的话，就需要分为三步：

* 初始化数据



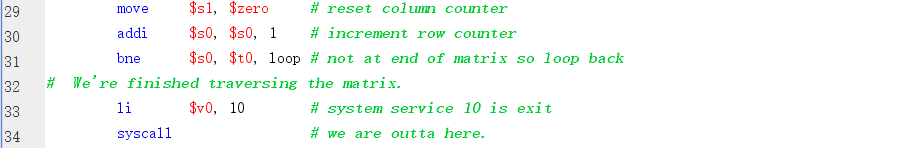
$t0和$t1是总的行列数，$s0和$s1是当前赋值的行列数，$t2就是要赋的值。

* 为一行矩阵赋值



19~23行为当前矩阵元素对应的内存赋值，24,27行更新数据，28行判断这一行矩阵是否已完全赋值。

* 然后为下一行矩阵



当一行矩阵已赋完值后，更新数据，然后重新跳回到loop处为下一行矩阵赋值，直到整个矩阵都赋完值。