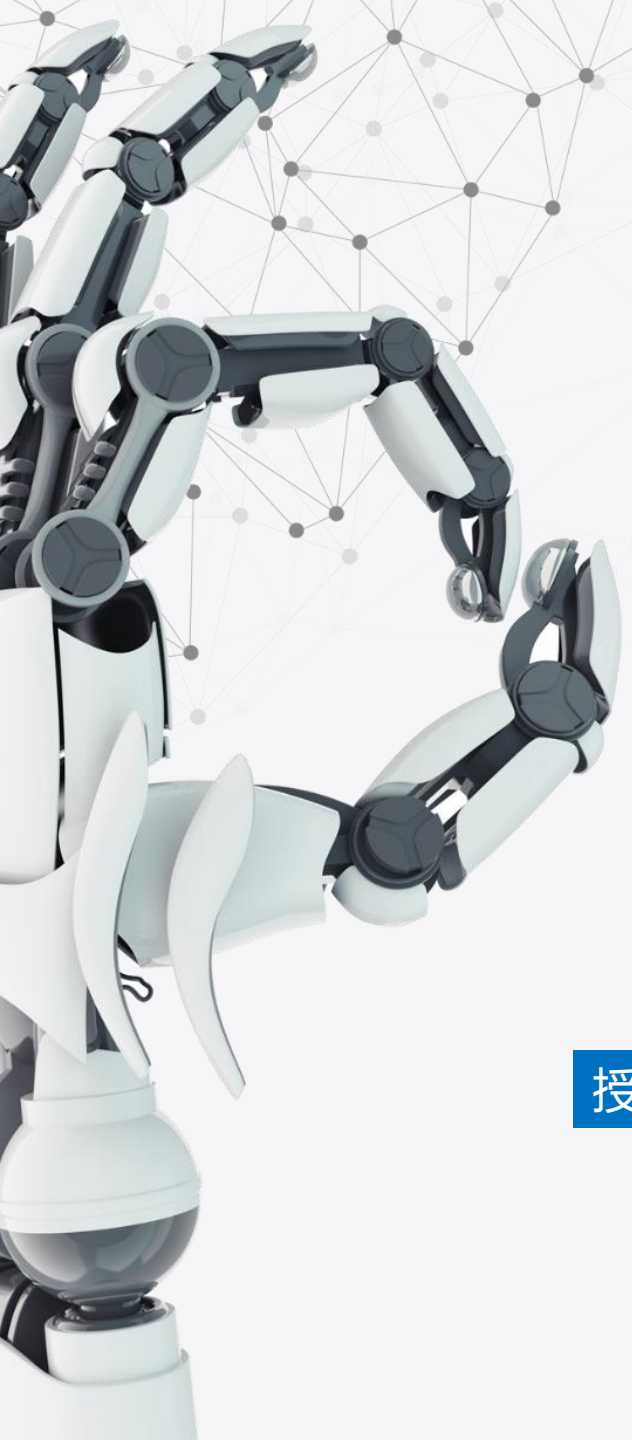


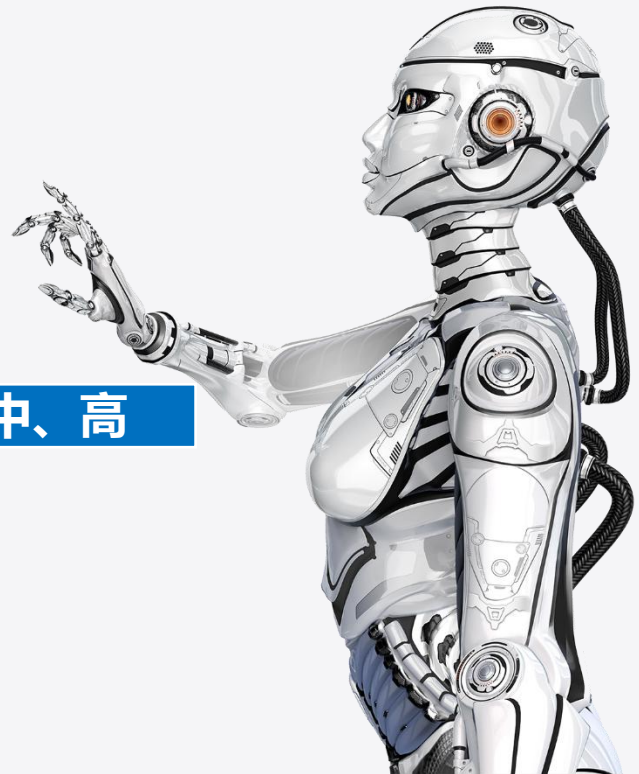
结构体

授课日期：

授课时间：

授课教师：

考试所涉及级别：初、中、高



目 录
CONTENT

01

动态数组

您的内容打在这里或者通过复制您的文本后，在此框中选择粘贴，并选择只保留文字。

> C++ STL（Standard Template Library，标准模板库）是惠普实验室开发的一系列软件的统称，是一套功能强大的 C++ 模板类。STL 实现了集合、映射表、栈、队列等数据结构和排序、查找等算法。

学习 STL 有哪些意义？

- 我们可以很方便地调用标准库来减少我们的代码量。

STL容器 对最常用的数据结构提供了支持。

- ① 顺序容器：动态数组 `vector` 、双端队列 `deque` 、双向链表 `list` 等。
- ② 关联容器：集合 `set` 、多重集合 `multiset` 、映射 `map` 、多重映射 `multimap` 等。

首先：为什么需要动态定义数组呢?
这是因为，很多情况下，在预编译过程阶段，数组的长度是不能预先知道的，必须在程序运行时动态的给出
但是问题是，c++要求定义数组时，必须明确给定数组的大小，要不然编译通不过

如： int Array[5];正确

```
int i=5;
int Array[i]; 错误 因为在编译阶段， 编译器并不知道 i 的值是多少
```

那么，我们该如何解决定义长度未知的数组呢?
答案是：new 动态定义数组

因为new 就是用来动态开辟空间的，所以当然可以用来开辟一个数组空间

这样，下面的语句：

```
int size=50;
int *p=new int[size]; 是正确的
```

动态数组的定义

定义：一个名为 vec 的储存 T 类型数据的动态数组。
数组要储存的数据类型：T，它可以是 int 、 double 、 string 、或者其他自定义的数据类型等等。
初始的时候 vec 是空的。

```
vector<T> vec;
```

在最后插入元素
C++ 中通过 push_back() 函数在 vector 最后面插入一个新的元素。

```
#include <vector>
using namespace std;
int main() {
    vector<int> vec;  // 初始：[]
    vec.push_back(1); // 当前：[1]
    vec.push_back(2); // 当前：[1, 2]
    vec.push_back(3); // 当前：[1, 2, 3]
    return 0;
}
```

```
#include <vector>
using namespace std;
int main() {
    vector<int> vec;  // 初始：[]
    vec.push_back(1); // 当前：[1]
    vec.push_back(2); // 当前：[1, 2]
    vec.push_back(3); // 当前：[1, 2, 3]
    vec.pop_back(); // 当前：[1, 2]
    vec.pop_back(); // 当前：[1]
    return 0;
}
```

从末尾删除元素
C++ 中通过 pop_back() 函数删除 vector 末尾的一个元素。

```
#include <vector>
#include <iostream>
using namespace std;
int main() {
    vector<int> vec;  // 初始：[]
    vec.push_back(1); // 当前：[1]
    vec.push_back(2); // 当前：[1, 2]
    vec.push_back(3); // 当前：[1, 2, 3]
    for (int i = 0; i < vec.size(); i++) {
        cout << vec[i] << endl;
    }
    return 0;
}
```

获取长度并且访问元素
通过 size() 函数获取 vector 的长度

通过 [] 操作直接访问 vector 中的元素，下标访问元素的方式和数组是一样的。

修改元素

C++ 中修改 vector 中某个元素很简单，只需要用=给它赋值就好了，比如 vec[1] = 3

```
#include <vector>
#include <iostream>
using namespace std;
int main() {
    vector<int> vec;  // 初始：[]
    vec.push_back(1); // 当前：[1]
    vec.push_back(2); // 当前：[1, 2]
    vec.push_back(3); // 当前：[1, 2, 3]
    vec[1] = 3; // 当前：[1, 3, 3]
    vec[2] = 2; // 当前：[1, 3, 2]
    for (int i = 0; i < vec.size(); i++) {
        cout << vec[i] << endl;
    }
    return 0;
}
```

C++ `vector` 函数总结：

函数	功能
push_back	在末尾加入一个元素
pop_back	在末尾弹出一个元素
size	获取长度
clear	清空
empty	判断容器是否为空

清空

C++ 中都只需要调用 `clear()` 函数就可清空 `vector` 。

```
vector<int> vec;
vec.clear();
```

C++ 中 `vector` 的 `clear()` 只是清空 `vector` ，并不会释放开辟的内存，用下面的方法可以释放 `vec` 的内存：

```
vector<int>().swap(vec);
```

复制

比如：

```
    vector<int> vec;
    for (int i = 2; i <= 10; i += 2) {
        vec.push_back(i);
    }
    vec.clear();
    cout << vec[0] << " " << vec[1] << endl;
    vector<int>().swap(vec);
    cout << vec[0] << " " << vec[1] << endl;
```

复制

小结：

调用 `clear()` 进行清理，如果此时打印 `vector[0]` ，会发现仍然输出之前 `vector` 所存的内容，但是如果调用 `empty()` 函数又会返回 $true$，这就是因为使用 `clear()` 清空内容，但是没有释放内存的原因。

数组 VS 动态数组

3. 数据的存储：

- ① 元素索引
  - 普通数组：对于一个长为 n 的数组来说，元素索引范围是 $0 \sim n-1$ 。
  - 动态数组：假设动态数组名为 v，元素索引范围是 $0 \sim v.size()-1$ 。

> 如果超过索引范围，去访问数组元素，会发生访问越界的问题。

- ② 元素的赋值
  - 普通数组：声明之后，可以直接通过索引给元素赋值；
  - 动态数组：声明之后，通过 `push_back()` 添加元素，之后才可以给已添加的元素进行值的修改。

- ③ `vector` 可以通过动态数组名，直接赋值

- ③ `vector` 可以通过动态数组名，直接赋值

复制

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> vec1;
    vector<int> vec2;
    for (int i = 2; i <= 10; i += 2) {
        vec1.push_back(i);
    }
    vec2 = vec1; // 将 vec1 中的元素值整体赋值给 vec2
    for (int i = 0; i < vec2.size(); i++) {
        cout << vec2[i] << endl;
    }
    return 0;
}
```

数组 VS 动态数组

1. 创建方式：

```
int a[1005];
vector<int> v;
```

2. 数组长度

- **普通数组** 的大小的一开始定义的时候就已经确定，在理论上 **不可扩容** 。
- **动态数组** 在程序运行过程中能够根据实际情况动态的 **改变数组大小**，主要通过 $push\_back$ 和 pop_back 两个操作来实现。

```
v.push_back(x); // 在动态数组 v 最末端增加一个值为 x 的元素
v.pop_back(); // 删除动态数组 v 最末端的元素
```

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> vec;
    //调用 push_back 函数，向 vec 这个动态数组中插入一些元素；
    for (int i = 2; i <= 10; i += 2) {
    vec.push_back(i);
}
    for (int i = 0; i < vec.size(); i++) {
    vec[i] += 1;
    cout << vec[i] << endl;
}
    vec.pop_back();
    cout << vec.size() << endl;
    //调用 clear 函数来清空动态数组；
//清空后 vec 的大小变为 0。
    vec.clear();
    cout << vec.size() << endl;
    //这条语句通常用来回收 锯齿数组 (二维动态数组) 中的某一行元素的内存。
    vector<int>().swap(vec);
    return 0;
}
```

02
动态数组进阶

小弹珠弹弹

动态数组排序

回忆一下普通数组的排序过程

```
int arr[5] = {4, 7, 1, 4, 6};
sort(arr, arr + 5);
```

对 `vector` 排序可以这样：

```
sort(vec.begin(), vec.end());
```

- `begin` 函数获得的是动态数组中首元素的迭代器；
- `end` 函数获得的是动态数组的尾后迭代器。

> 如果你不了解迭代器，只需要先记住用法，在 A7 阶段我们将会具体学习。

如果你只想对 `vector` **前三个元素** 排序，也可以这样写：

```
sort(vec.begin(), vec.begin() + 3);
```

想一想：如果要对第 2 个元素到第 5 个元素进行排序，应该怎么写呢？

用动态数组储存自定义类型的数据

动态数组不仅仅可以储存基本的数据类型，还能储存 **自定义数据类型** ，比如结构体。

```
struct Student {
    string name;  // 名字
    int age;      // 年龄
};
int main() {
    vector<Student> class1;  // 班级
    Student stu;       // 学生
    stu.name = "xiaohong";
    stu.age = 12;
    class1.push_back(stu);
    return 0;
}
```

构造函数

我们知道可以通过 `push_back()` 来向动态数组添加一个元素。如果我们需要一个长度为 n 的，全是 1 的动态数组。我们可以像下面这样写。

```
int n = 10;
vector<int> vec;
for (int i = 0; i < n; i++) {
    vec.push_back(1);
}
```

应用 **构造函数**，可以在定义一个动态数组时，直接赋初始值。

```
int n = 10;
vector<int> vec(n, 1);
```

复制

调用构造函数：

- 第一个参数表示初始的动态数组的长度；
- 第二个参数表示初始的数组里面每个元素的值。

注意： 如果不传入第二个参数，那么初始的值都是 0。


```cpp
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
int main() {
    int n;
    cin >> n;
    //定义一个储存int类型数据的动态数组vec
    vector<int> vec;
    //我们用循环读入 nn 个整数，每次放入动态数组尾部
    for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    vec.push_back(x);
}
  //第一个参数传入首元素的迭代器vec.begin();
    //第二个参数传入尾后迭代器vec.end()。
    sort(vec.begin(), vec.end());
    for (int i = 0; i < n; i++) {
    cout << vec[i] << endl;
}
```

- 如果是对 **数组a** 进行排序：

```
sort(a + 起点下标, a + 终点下标 + 1)
```

- 如果对 `string` **变量s** 进行内部排序：

```
sort(s.begin(), s.end())
```

复制

- 如果对 `vector` **变量v** 进行排序：

```
sort(v.begin(), v.end())
```

练一练

现在我们有一个动态数组 `vector<double> vec` ，里面储存了 100 个浮点数。

现在要对 `vec[50]` 、 `vec[51]` 、…… `vec[79]` 进行从小到大排序。请你选出所有 **正确** 的排序代码。

请勾选所有符合题目要求的选项后再提交答案

- [ ] `sort(vec.begin() + 50, vec.begin() + 79);`
- [ ] `sort(vec.end() - 20, vec.end() - 50);`
- [ ] `sort(vec + 50, vec + 79);`
- [ ] `sort(vec.begin() + 50, vec.end() - 20);` 对
- [ ] `sort(vec.begin() + 50, vec.begin() + 80);` 对
- [ ] `sort(vec + 50, vec + 80);`

动态数组

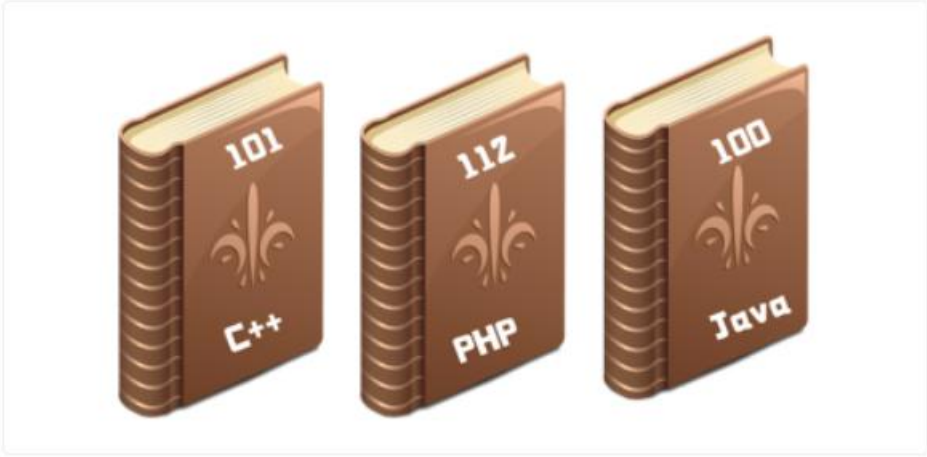
- 动态数组 vector 有以下优点：
- 可以动态改变数组大小，通常体现在 push_back() 和 pop_back()
- 随机访问方便，支持 [] 操作符直接读取相应下标的元素
- 节省空间

- 有以下缺点：
- 在 vector 内部进行插入、删除等操作效率较低
- 只能在 vector 的尾端进行 push 和 pop，不能在 vector 的头部进行 push 和 pop

运用

任务描述

蒜头君的书房里有 n 本书，每本书有一个编号和书名。现在请你用动态数组储存所有的书，并按照编号从大到小输出所有书名。



输入格式

第一行输入一个整数 n，代表要放的书本数目。

接下来每行输入一个整数 id 代表该书的编号，以及一个字符串 $name$ 代表书名。

输出格式

输出有一行，排序后的书名，书名之间用空格隔开。

book1.cpp

```cpp
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
struct Book {
    int id;
    string name;
};
bool cmp(Book p1, Book p2) {
    return p1.id > p2.id;
}
int main() {
    int n;
    cin >> n;
vector<Book> books;
    for (int i = 0; i < n; i++) {
        Book t;
        cin >> t.id >> t.name;
        books.push_back(t);
    }
    sort(books.begin(),books.end(),cmp);
    for (int i = 0; i < n; i++) {
        cout << books[i].name << endl;
    }
    return 0;
}
```

我们要定义一个长度为 n 的，全是 0 的动态数组 `x`，下面哪种方法是 **错误** 的。

请勾选所有符合题目要求的选项后再提交答案

- [ ] `vector<int> x(n, 0);`
- [ ]
```
vector<int> x;
for (int i = 0; i < n; ++i) {
    x.push_back(0);
}
```
- [ ] `vector<int> x(0, n);`
- [ ] `vector<int> x(n);`

✅ `vector<int> x(0, n);`

第一个参数是长度，第二个参数是初始的值，这里写反了。

❌ `vector<int> x(n);`

默认的初始值为 0，所以是对的。

❌ `vector<int> x(n, 0);`

❌

```
vector<int> x;
for (int i = 0; i < n; ++i) {
    x.push_back(0);
}
```

选择正确的排序方法

本节内容计 12 分，每次错误减 4 分

现在我们有一个动态数组 `vector<double> vec` ，里面储存了 100 个浮点数。

现在要对 `vec[50]` 、 `vec[51]` 、..... `vec[79]` 进行从小到大排序。请你选出所有 **正确** 的排序代码。

请勾选所有符合题目要求的选项后再提交答案

- [ ] `sort(vec + 50, vec + 79);`
- [ ] `sort(vec.begin() + 50, vec.end() - 20);`
- [ ] `sort(vec.begin() + 50, vec.begin() + 80);`
- [ ] `sort(vec.begin() + 50, vec.begin() + 79);`
- [x] `sort(vec + 50, vec + 80);`
- [ ] `sort(vec.end() - 20, vec.end() - 50);`

动态数组 `vector` 有以下优点：

- 可以动态改变数组大小，通常体现在 `push_back()` 和 `pop_back()`
- 随机访问方便，支持 `[]` 操作符直接读取相应下标的元素
- 节省空间

有以下缺点：

- 在 `vector` 内部进行插入、删除等操作效率较低
- 只能在 `vector` 的尾端进行 $push$ 和 pop，不能在 `vector` 的头部进行 $push$ 和 pop

在未来的学习过程中，我们还会遇到很多动态数组的应用：

- vector 和其他容器的结合

比如 `vector` 和 `map` 的结合：

```
map<int, vector<int> > mp;
```

- 二维动态数组

在下一节课我们会学到二维动态数组，二维数组也有很多应用。

- 利用 vector 建立邻接表

在未来我们会学到树和图，可以用 vector 简化 **建图** 过程。

1000ms 131072K

学校的一年一度的合唱比赛又要开始了，同学们都积极踊跃的报名。

同学一个个按顺序报名，并加入到队伍末尾。老师为了维持队伍美观，时不时要当前队伍中的某个区间的同学按照身高从小到大排序。问最后同学的身高排列。

提示：请用动态数组完成哦。

输入格式

第一行包含一个正整数 q $(1 \leq q \leq 10^3)$，代表 q 次操作，同学报名或老师排序；

接下来 q 行每行首先输入一个整数 op 。

- 当 $op == 1$ 时，表示同学报名，接下来输入 1 个整数 h，表示当前报名的学生身高。
- 当 $op == 2$ 时，表示老师要进行排序，接下来输入两个整数 x 、 y ，表示老师要对当前第 x 个同学到第 y 个同学进行排序。

输出格式

输出共一行，最后同学的身高排列。同学的身高之间用一个空格隔开。

格式说明

输出时每行末尾的多余空格，不影响答案正确性

样例输入1

```
5
1 3
1 2
1 1
2 2 3
1 4
```

样例输出1

```
3 1 2 4
```

样例输入2

```
6
1 3
1 2
1 1
2 1 2
1 4
2 3 4
```

样例输出2

```
2 3 1 4
```

C++ 语言

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <cstdio>
using namespace std;
int main()
{
    int q, op, h, x, y;
    vector<int> v;
    cin >> q;
    while (q--) {
        cin >> op;
        if (op == 1) {
            cin >> h;
            v.push_back(h);
        } else {
            cin >> x >> y;
            sort(v.begin() + x - 1, v.begin() + y);
        }
    }
    for (int i = 0; i < v.size(); i++) {
        cout << v[i] << " ";
    }
    cout << endl;
    return 0;
}
```

双重间谍

- X国的情报委员收到一份可靠的信息，信息表明Y国将派间谍去窃取X国的机密文件。X国指挥官手中有两份名单列表，一份是Y国派往X国的间谍名单列表，另一份是X国以前派往Y国的间谍名单列表。这两份名单列表可能有些重叠。因为间谍可能同时扮演两个角色，称之为“双重间谍”。因此，Y国可以把双重间谍送回X国。很明显，这对X国是有利的，因为双重间谍可以把Y国的机密文件带回，而不必担心被Y国边境拘留。所以指挥官决定抓住由Y国派出的间谍，让普通人和双重间谍进入。那么你能确定指挥官需要抓捕的间谍名单吗？输入： 有几个测试用例。每个测试用例都包含4部分。第1部分包含3个正整数A 、B 、C ， A 是进入边境的人数， B 是Y国将派出的间谍人数， C 是X国以前派到Y国的间谍人数。第2部分包含A 个字符串，为进入边境的人员名单。第3部分包含B 个字符串，为由Y国派出的间谍名单。第4部分包含C 个字符串，即双重间谍的名单。每个测试用例后都有一个空白行。在一份名表中不会有任何名字重复，如果有重复的名字出现在两份名单列表中，则表示同一个人。输出： 输出指挥官抓捕的间谍名单（按列表B的出现顺序）。如果不应捕获任何人，则输出“No enemy spy”。

输入输出样例

```
8 4 3
Zhao Qian Sun Li Zhou Wu Zheng Wang
Zhao Qian Sun Li
Zhao Zhou Zheng
2 2 2
Zhao Qian
Zhao Qian
Zhao Qian


```

输出样例

```
Qian Sun Li
No enemy spy
```

```
#include<iostream>
#include<vector>
#include<algorithm>
#include<string>
using namespace std;
int main(){
	int a,b,c;
	string s;
	vector<string> x,y,z,ans;
	while(cin >> a >> b >> c){
		x.clear();
		y.clear();
		z.clear();
		ans.clear();

		for(int i  = 0 ;i< a;i++){ //一次for循环拿下入境人数
			cin >> s;
			x.push_back(s);
		}

		for(int i = 0;i<b;i++){//这个for循环拿下派遣的间谍
			cin >> s;
			y.push_back(s);
		}

		for(int i = 0;i<c;i++){//这个fpr循环拿下双重间谍
			cin >> s;
			z.push_back(s);
		}


		for(int i = 0;i<b;i++){
			if(find(x.begin(),x.end(),y[i]) != x.end()){
				if(find(z.begin(),z.end(),y[i]) == z.end() ){
					ans.push_back(y[i]);
				}
			}
		}

		if(!ans.size()){
			cout << "No enemy spy\n";
		}else{
			for(int i =0;i<ans.size();i++){
				if(i != 0)
					cout << " ";
				cout << ans[i];
			}
			cout << endl;
		}


	}
	return 0;
}
```

指间宇宙
一编程高手一

03

二维动态数组

小弹珠弹弹

但是思考一个问题如果我们希望有一个数组，它的长度是可以变化的，该怎么做呢?

也就是说我不想再一开始就定义一个长度为 100 的数组了，我希望我有 50 个元素的时候数组长度就是 50，100 个元素的时候数组长度就是 100

其实我们是学习过这种性质的类型的，思考一下是什么类型?

我们学习过的 `string` 类型其实就可以满足这个条件，我们说 `string` 它是一个变量类型，但它本质上是个字符数组

但是 `string` 只允许访问下标 $0 \sim size() - 1$，而这个 $size()$ 是会动态发生变化的

所以其实 `string` 就像是一个 `动态数组`，可以依据里面存放的元素不同，来动态的变化数组长度

当我们需要用到多个 vector 的时候，会用到 vector 的数组。

```
vector<int> a[10];
for (int i = 0; i < 10; i++) {
    for (int j = 0; j <= i; j++) {
        a[i].push_back(j);
    }
}
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < a[i].size(); j++) {
        cout << a[i][j] << " ";
    }
    cout << endl;
}
```

打印的结果如下：

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4 5 6
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9
```

用法和普通数组一样，只不过 a 的每一个元素都是一个 vector， a 就相当于第二维长度可以变化的二维数组。

打印的结果如下：

主要是记住一维和二维数组的定义以及区别，它们初始化，和添加删除元素是有点不太一样的。

一维数组

```
vector<int> arr;
arr.size();
arr.begin();
arr.insert();
arr.push_back();
arr.end();
```

二维数组

```
vector<vector<int>> A;//错误的定义方式
vector<vector<int> > A;//正缺的定义方式
vector<vector<int> > v;//注意>和>之间的空格。
```

插入

```
//正确的插入方式
vector<vector<int> > A;
//A.push_back里必须是vector
vector<int> B;
B.push_back(0);
B.push_back(1);
B.push_back(2);
A.push_back(B);
B.clear();
B.push_back(3);
B.push_back(4);
B.push_back(5);
A.push_back(B);
```

复制

错误

```
//错误的插入方式
vector<vector<int> > A;
A[0].push_back(0);
A[0].push_back(1);
A[0].push_back(2);
A[1].push_back(3);
A[1].push_back(4);
A[1].push_back(5);
```

长度

```
//vector<vector<int> >A中的vector元素的个数
len = A.size();
//vector<vector<int> >A中第i个vector元素的长度
len = A[i].size();
```

访问二维vector的元素的四种方式

- 如果指定外层和内层向量的大小，就可用operator[]进行读和写；如果只指定外层向量大小，就能用push_back()函数进行写，不能用operator[]进行读和写。

push_back()函数进行写，不能用operator[]进行读和写。

1）指定外层vector大小

可用push_back函数进行初始化：

```
v.resize(3);
v[1].push_back(9);
```

2）遍历指定内层vector大小

提前设定好每行vector的大小，就可用operator[]访问，如下：

```
for(int i=0;i<3;i++)
    v[i].resize(3);
```

3）一次指定内外层vector大小

```
v.resize(n, vector<int>(m));
```

4）直接使用二维数组的形式访问

```
//根据前面的插入，可知输出5。
printf("%d\n", A[1][2]);
```

任务描述

- 指间君的书房里有3个书架，现在他要把n本书放到书架上，每本书会告诉你书名name,并要放到第k个书架上。请用动态数组存放所有的书，并输出最终三个书架上的结果。
- 输入格式
- 第一行输入一个整数n,代表要放的书本数目。
- 接下来每行输入一个字符串name代表书名，以及一个整数k(1≤k≤3)代表该书要放入的书架。
- 输出格式
- 输出有三行，每个书架上的书名，书名之间用空格隔开。

这一节已经完成了，点击 **运行** 输入下面这组数据查看效果。

复制

```
10
C++ 3
PHP 2
JavaScript 1
Go 2
Perl 1
R 1
C 3
Java 3
Python 2
C# 3
```

vector.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main() {
    int n;
    cin >> n;
vector<string> shelf[4];

    for (int i = 0; i < n; i++) {
        string name;
        int k;
        cin >> name >> k;
shelf[k].push_back(name);
    }
    for (int i = 1; i <= 3; i++) {
    for (int j = 0; j < shelf[i].size(); j++) {
        cout << shelf[i][j] << " ";
    }
    cout << endl;
}

```