copy

需要头文件`iterator`

```
copy(iVec.begin(),iVec.end(),ostream_iterator<int>(cout," "));
```

如果要把一个序列（sequence）拷贝到一个容器（container）中去，通常用std::copy算法，代码如下：

```
std::copy(start, end, std::back_inserter(container));
```

这里，start和end是输入序列（假设有N个元素）的迭代器（iterator），container是一个容器，该容器的接口包含函数push_back。假设container开始是空的，那么copy完毕后它就包含N个元素，并且顺序与原来队列中的元素顺序一样。标准库提供的back_inserter模板函数很方便，因为它为container返回一个back_insert_iterator迭代器，这样，复制的元素都被追加到container的末尾了。

现在假设container开始非空（例如：container必须在循环中反复被使用好几次）。那么，要达到原来的目标，必须先调用clear函数然后才能插入新序列。这会导致旧的元素对象被析构，新添加进来的被构造。不仅如此，container自身使用的动态内存也会被释放然后又创建，就像list，map，set的节点。某些vector的实现在调用clear的时候甚至会释放所有内存。

通常，考虑到在一个已有的元素上直接copy覆盖更高效。刻意这样做：

```
std::copy(start, end, container.begin());
```

在这里你在container的头部执行了copy－over（覆盖赋值）操作，但是，如果container的大小小于输入序列的长度N的话，这段代码会导致崩溃（crash）。

```
eg1:
int a[3] = {1, 2, 3};
int b[3];
std::copy(a, a+3, b);
for(int j=0; j<3; j++)
    cout<< b[j] << endl;

eg2:
vector temp(3);
int a[3] = {1, 2, 3};
std::copy(a, a+3, &temp.front());
for(int j=0; j<3; j++)
    cout<< temp[j] << endl;
```

copy只负责复制，不负责申请空间，所以复制前必须有足够的空间