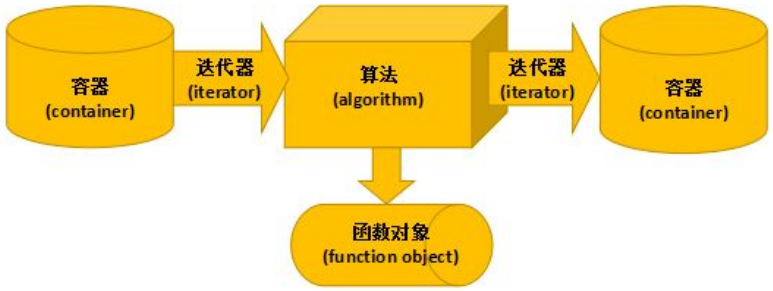
# C++语言程序设计

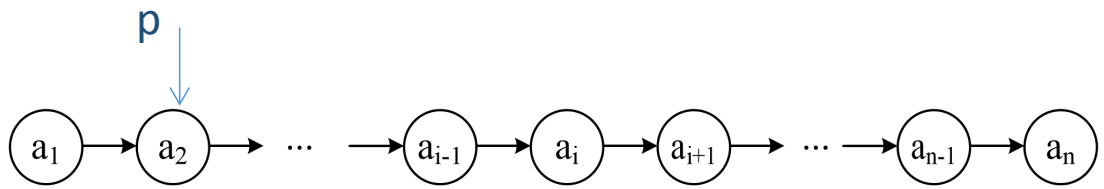
贺利坚 主讲

## 迭代器

# 迭代器及其分类




- 迭代器是算法和容器的“中间人”。
- 迭代器对存储在容器中的元素序列进行遍历，提供的访问容器中每个元素的方法。
- 指针是迭代器中的一种。
- 迭代器是泛化的指针，提供了类似指针的操作（诸如++、*、->运算符）
- 迭代器不仅仅是指针
  - 迭代器更为抽象，可以指向容器中的一个位置，通过迭代器访问这个位置的元素，而不必关心这个位置对应的是真正的物理地址。




**输入迭代器 (Input Iterator)**

可以用来从序列中读取数据

**输出迭代器 (Output Iterator)**

允许向序列中写入数据

**前向迭代器(Forward Iterator)**

既是输入迭代器又是输出迭代器，并且可以对序列进行单向的遍历

**双向迭代器(Bidirectional Iterator)**

与前向迭代器相似，但是在两个方向上都可以对数据遍历

**随机访问迭代器(Random Access Iterator)**

也是双向迭代器，但能够在序列中的任意两个位置之间进行跳转，如指针、使用vector的begin()、end()函数得到的迭代器

# 迭代器的操作

输入迭代器：
p1==p2; p1!=p2;
*p1;
p1->m;
*p1++;

输出迭代器：
*p1=t;
*p1++=t;

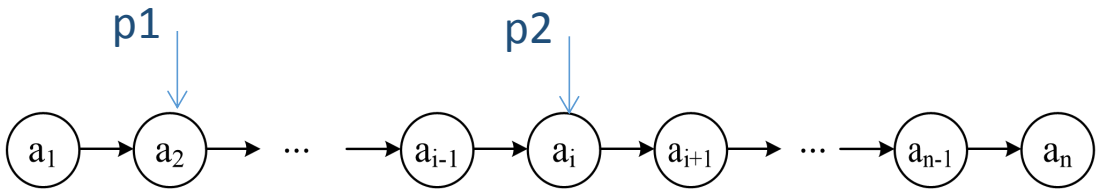
p1++的返回值不确定

前向迭代器更加明确：
*p1;
p1++;

双向迭代器新增：
--p1;p1--;

随机访问迭代器新增：
p1+=n; p1-=n;
p1+n; n+p1; p1-n;
p1-p2;
p1比较运算符p2;
p1[n];

- 两个迭代器表示一个区间：[p1, p2)
- STL算法常以迭代器的区间作为输入，传递输入数据
- 例：
  transform(p1,p2,...);
  transform(s.begin(),
  s.end(),...)
- 区间包含p1，但不包含p2




输入迭代器
(Input Iterator)

输出迭代器
(Output Iterator)

前向迭代器
(Forward Iterator)

双向迭代器
(Bidirectional Iterator)

随机访问迭代器
(Random Access Iterator)

概念

子概念

# 输入流迭代器和输出流迭代器

- 输入流、输出流
  - 标准类：istream、ostream
  - 实例：cin、cout
- 输入流迭代器
  - template<class T> istream_iterator<T>
  - 要求：T类型重载了运算符 >>
  - 以输入流为参数构造：

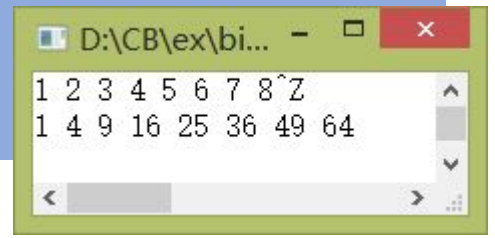
  istream_iterator<int>(cin);

  istream_iterator<int>();//指向输入流结束位置
- 输出流迭代器
  - template<class T> ostream_iterator<T>
  - 要求：T类型重载了运算符 <<
  - 以输出流为参数构造：

  ostream_iterator<int>(cout);

  ostream_iterator<int>(cout, 分隔符);

```
#include <iterator>
#include <iostream>
#include <algorithm>
using namespace std;
//求平方的函数
double square(double x)
{
   return x * x;
}
int main()
{
   transform(istream_iterator<double>(cin),
                        istream_iterator<double>(),
                        ostream_iterator<double>(cout, " "),
                        square);
   cout << endl;
   return 0;
}
```

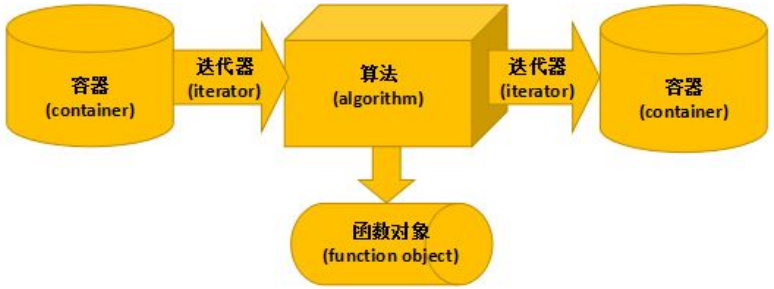


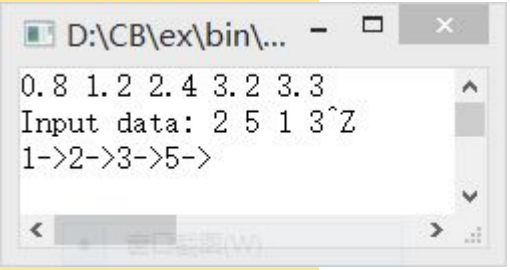
## 再例：

```
#include <algorithm>
#include <iterator>
#include <vector>
#include <iostream>
using namespace std;
template <class T, class InputIterator, class OutputIterator>
void mySort(InputIterator first, InputIterator last, OutputIterator result)
{
   vector<T> s;
   for (; first != last; ++first)
      s.push_back(*first);
   sort(s.begin(), s.end());
   copy(s.begin(), s.end(), result);
}
```

```
int main()
{
   double a[5] = { 1.2, 2.4, 0.8, 3.3, 3.2 };
   mySort<double>(a, a + 5, ostream_iterator<double>(cout, " "));
   cout << endl;
   cout<<"Input data: ";
   mySort<int>(istream_iterator<int>(cin),
               istream_iterator<int>(),
               ostream_iterator<int>(cout, "->"));
   cout << endl;
   return 0;
}
```

- 迭代器是算法和容器的“中间人”。



# 迭代器的辅助函数

- 对pos执行n次自增操作
  - void advance(Iterator pos,int n);

    //不检查迭代器是否超过end()
- 计算两个迭代器first和last的距离
  - int distance(Iterator first, Iterator last)

    //对first执行多少次“++”操作后

    //能够使得first == last

```
#include <iostream>
#include <list>
#include <algorithm>
#include <iterator>
using namespace std;
int main()
{
   list<int> coll;
   for(int i = 1; i <= 9; ++i)
   {
      coll.push_back(i);
   }
   list<int>::iterator pos=coll.begin();
   advance(pos,2);
   cout<<"value: "<< *pos<<endl;
   pos = find(coll.begin(), coll.end(), 6);
   if(pos != coll.end())
   {
      cout << "Distance: " << distance(coll.begin(), pos);
      cout << endl;
   }
   return 0;
}
```