lecture6

what：Iterators and pointers
what more:Iterator Basics Iterators. pointers and memory

引言：container Interface
container.begin()-->指向第一个元素
container.end()-->指向最后一个元素

> 2025Fall-06-Iterators, p.28
>
> > end() never points to an element!

interface:
auto it=c.begin()
++it
auto & elem=* it

Iterator Interface

```
// Copy construction
auto it = c.begin();
```

```
// Increment iterator forward
++it;
```

```
// Dereference iterator -- undefined if it == end()
auto& elem = *it;
```

```
// Equality: are we in the same spot?
if (it == c.end()) ...
```

区分it++与++it
++it:本质是Iterator&operator++ 返回的是Iterator&(就是直接返回引用本身)

it++:本质上先拷贝之前的operator(int),在移动，然后删除，返回Iterator（会创建迭代器副本）
综上：++it更好，不用拷贝副本，效率更高，尽管现代编译器会把it++优化成++it,但本质不变，多用++it

```
for(int i=0;i<n;i++)//++i

int a=i++; //a=i
int b=++i;b=i+1
```

++it avoids making an unnecessary copy

```
// Prefix form - ++it
// Increments it and returns a reference to same object
Iterator& operator++();
```

```
// Postfix form - it++
// Increments it and returns a copy of the old value
Iterator operator++(int);
```

Remember: an iterator is a fully-fledged object, so it's often more expensive to copy than, say, an **int**

functionality:
Input/output 输出端：只往前走（stream）
forward:前向：支持multiple遍历
Bidirectional：双向：++it与--it(list与set)
Random access：随机 it[n] it+5(vector,deque)

Input/output 输出端：只往前走（stream）

Input Iterators

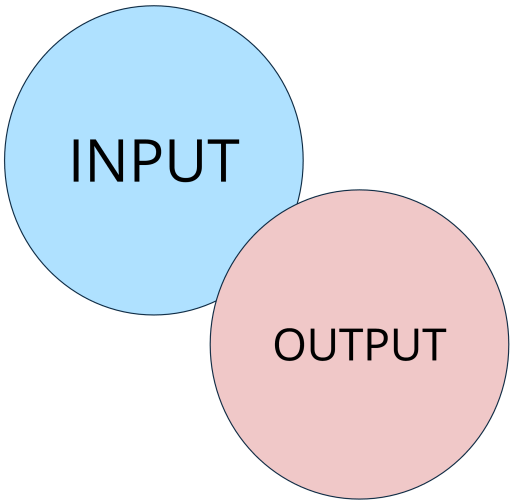
- Most basic kind of iterator
- Allows us to read elements

```
auto elem = *it;
```

Output Iterator

Allows us to write elements

```
*it = elem;
```



Vivid Venn Diagram of Vexing Iterators

forward:前向：支持multiple遍历

Forward Iterator

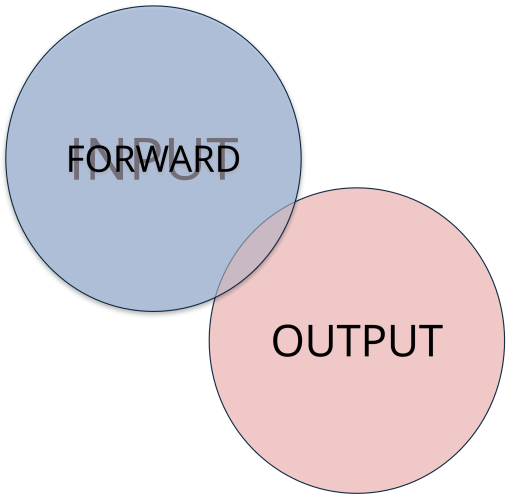
- An input iterator that allows us to make multiple passes
- All STL container iterators fall here

Multi-pass guarantee

```
it1 == it2
```

↓

```
++it1 == ++it2
```



What kind of data structure might not want a multi-pass iterator?

Streams!!!

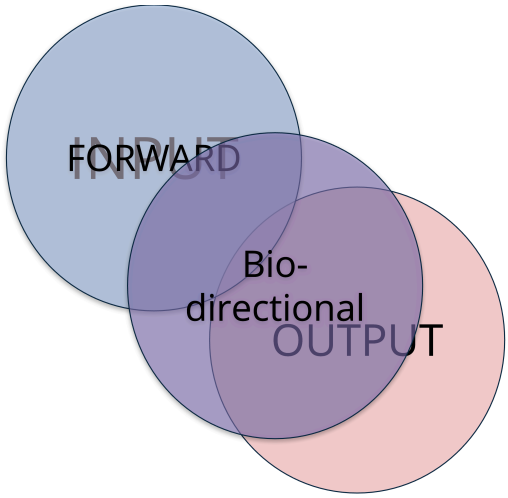
Bidirectional：双向：++it与--it(list与set)

Bidirectional Iterators

- Allows us to move forwards *and* backwards
- std:map, std::set

```
auto it = m.end();

// Get last element
--it;
auto& elem = *it;
```



Vivid Venn Diagram of Vexing Iterators

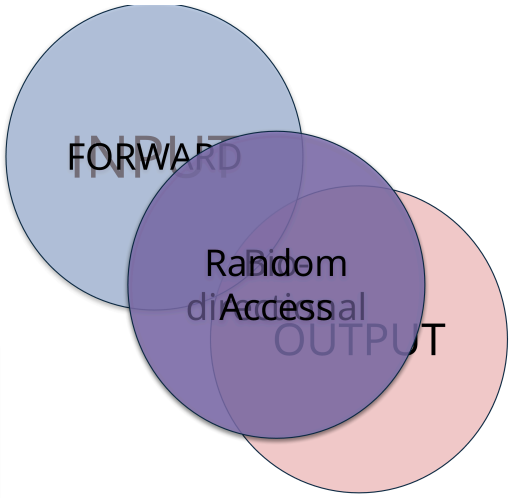
Random access：随机 it[n] it+5(vector,deque)

Random Access Iterators

- Allows us to quickly skip forward and backward
- std::vector, std::deque

```
auto it2 = it + 5; // 5 ahead
auto it3 = it2 - 2; // 2 back

// Get 3rd element
auto& second = *(it + 2);
auto& second = it[2];
```



Vivid Venn Diagram of Vexing Iterators

给迭代器分类有用吗（matter）
ex:sort()使用快排算法，需要这个迭代器是可以随机访问，便于跳转，所以必须it是random的，所以vector和deque可以用sort，map不能

Why does it matter?

As we'll soon see, some algorithms require a certain iterator type!

```
std::vector<int> vec{1,5,3,4};
std::sort(vec.begin(), vec.end());
// ✅ begin/end are random access

std::unordered_set<int> set {1,5,3,4};
std::sort(set.begin(), set.end());
// ❌ begin/end are bidirectional
```

Why have multiple iterator types?

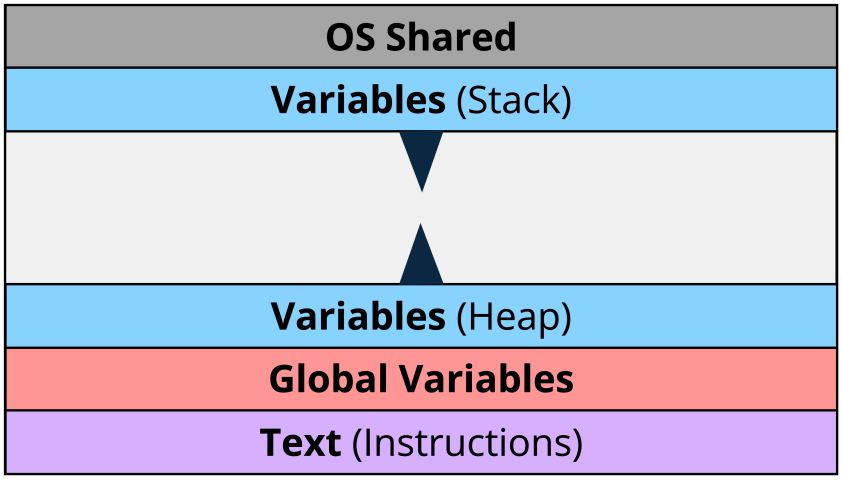
- **Goal:** provide a uniform abstraction over all containers
- **Caveat:** the way that a container is implemented affects how you iterate through it
  - Skipping ahead 5 steps (random access) is a lot easier/faster when you have a sequence container (vector, deque) than associative (map, set)
  - C++ generally avoids providing you with slow methods by design, so that's why you can't do random access on a map::iterator

pointers and memory
memory basics

OS shared-->内核
variables(stack)-->栈（局部变量，函数上下文）
variables(Heap)-->堆（主动申请的内存malloc new）
Global variables-->数据段（全局，静态变量）
Text(Instructions)-->只读(代码段)

Memory Basics

- Every variable lives somewhere in memory
- All the places something could live form the **address space**



关于常说的堆/栈溢出：
从这个图看，stack和heap的文件内存是有限的，如果分配不合理，会出现两块区域相遇，那就是溢出了，出现严重问题，需要注意

关于pointer：指针是一种特殊的迭代器（随机）

T* is the backing type for vector<T>::iterator

```
template <typename T>
class vector {
    using iterator = T*;

    // Implementation details...
};
```

In the real STL implementation, the actual type is not T*.
But for all intents and purposes, you can think of it this way.

What we covered

- Iterator Basics
  - An iterator allows us to step forward through a container
- Iterator Types
  - Input, Output, Forward, Bidirectional, Random Access
- Pointers and Memory
  - A pointer points to an arbitrary C++ object in memory
  - Pointers and iterators have the same interface