ساختمان داده ها

پیمایشگرها (Iterators)

مدرس: غیاثی‌شیرازی
دانشگاه فردوسی مشهد

چرا پیمایشگر؟

- یک نیازمندی مشترک برای ساختمان های داده، انجام عملی مشترک بر روی تمام اشیاء داده ذخیره شده در آن است.
- معمولا مرتبه زمانی رسیدن از عنصر فعلی به عنصر بعدی در پیمایشگرها کمتر از دسترسی مستقیم به یک عنصر است.

for i=1 to n
print L[i] $\Theta(i)$

$\sum_{i=1}^{n} i \in \Theta(n^2)$

for (itr = l.begin(); itr != l.end(); itr++) $\Theta(1)$
print(*itr)

$\Theta(n)$

پیمایشگرها در جاوا

- در جاوا هر کلاسی که می خواهد قابلیت پیمایش عناصر خود را در اختیار استفاده کننده قرار دهد، لازم است که رابط Iterable<Item> را پیاده سازی نماید.
- با فراخوانی متود iterator() می توان یک شیء از نوع Iterator<Item> دریافت کرد که می تواند بر روی عناصر کلاس که از نوع Item هستند، عمل پیمایش را انجام دهد.

```
public interface Iterable<Item>{
    Iterator<Item> iterator();
}
```

پیمایشگرها در جاوا

- کلاس Iterator<Item> امکان پیمایش بر روی عناصر کلاس اصلی را فراهم می کند.

```
public interface Iterator<Item>
{
    boolean hasNext();
    Item next();
    void remove(); // usually not implemented
}
```

در لحظه شروع، مکان ما یکی قبل از اولین عضو است.

پیمایشگرها در جاوا

- استفاده از پیمایشگرها (فرم shorthand)

```
for (String s : stack)
        StdOut.println(s);
```

استفاده از پیمایشگرها (فرم longhand)

```
Iterator<String> i = stack.iterator();
while (i.hasNext()){
        String s = i.next();
        StdOut.println(s);
}
```

پیمایشگرها در C++

- در زبان C++ قانون خاصی برای پیاده سازی پیمایشگرها وجود ندارد. اما STL طبقه بندی خود را از پیمایشگرها دارد.

- منبع اسلاید بعد:

http://www.cplusplus.com/reference/iterator/

category				properties	valid expressions
all categories				*copy-constructible*, *copy-assignable* and *destructible*	`X b(a);` `b = a;`
				Can be incremented	`++a` `a++`
Random Access	Bidirectional	Forward	Input	Supports equality/inequality comparisons	`a == b` `a != b`
				Can be dereferenced as an *rvalue*	`*a` `a->m`
			Output	Can be dereferenced as an *lvalue* (only for *mutable iterator types*)	`*a = t` `*a++ = t`
				default-constructible	`X a;` `X()`
				Multi-pass: neither dereferencing nor incrementing affects dereferenceability	`{ b=a; *a++;` `*b; }`

				Can be decremented	`--a`
					`a--`
					`*a--`

				Supports arithmetic operators + and −	`a + n`
					`n + a`
					`a - n`
					`a - b`

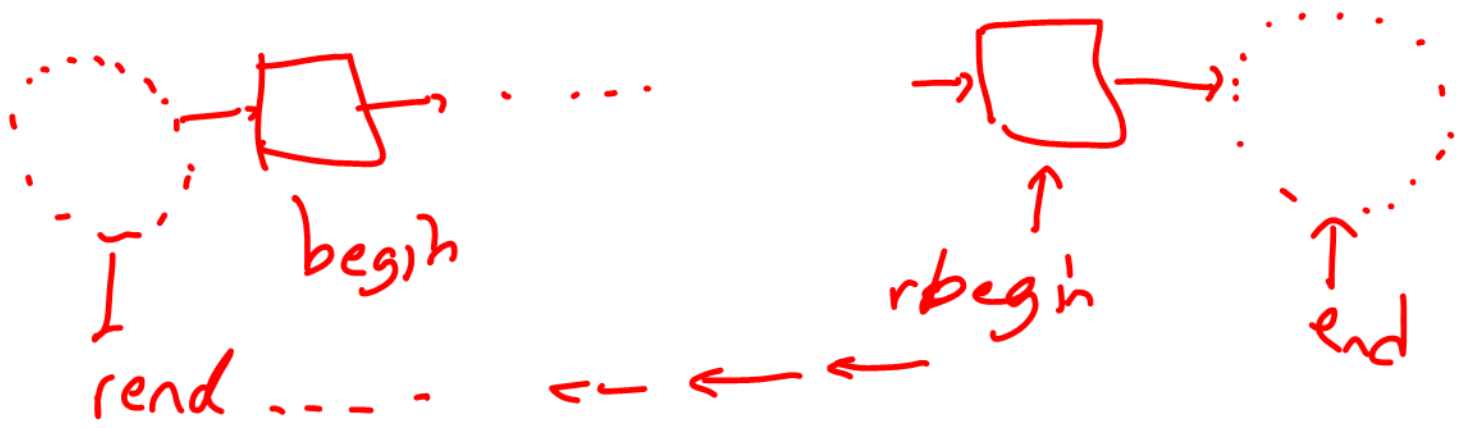
				Supports inequality comparisons (<, >, <= and >=) between iterators	`a < b`
					`a > b`
					`a <= b`
					`a >= b`

				Supports compound assignment operations += and −=	`a += n` `a -= n`
				Supports offset dereference operator (`[]`)	`a[n]`

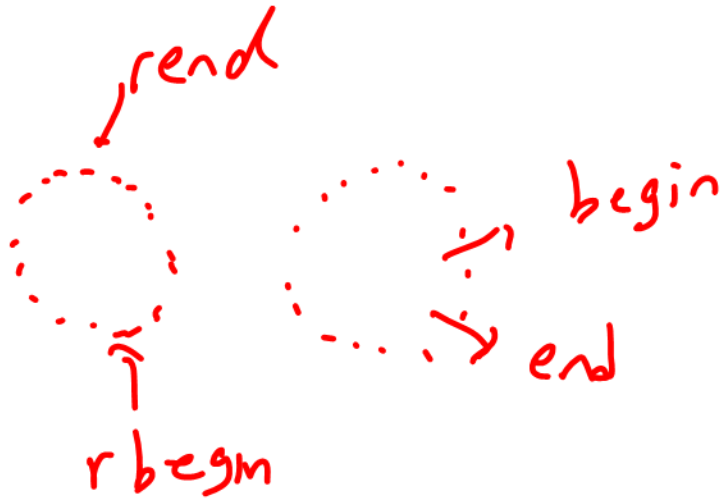
پیمایشگرها در C++

- در STL معمولا متودهای زیر از یک ساختمان داده، شیئی از نوع iterator برمی گردانند:
  - begin() – عنصر اول
  - end() – یکی بعد از آخرین عنصر
  - rbegin() – [reverse begin] آخرین عنصر
  - rend() – [reverse end] یکی قبل از اولین عنصر
- پیمایشگرهای begin() و rbegin() به اشیاء معتبری اشاره می کنند. اما پیمایشگرهای end() و rend() به یک عنصر بعد از آخرین عنصر و یک عنصر قبل از اولین عنصر اشاره می کنند.

در صورتی که ساختمان داده خالی نباشد.



اگر ساختمان داده خالی باشد



مثالی از پیمایش رو به جلو

```
std::list<int> mylist;

std::list<int>::iterator   it;

for (int i=1; i<=5; ++i)
        mylist.push_back(i);

for (it=mylist.begin(); it != mylist.end(); ++it)
        std::cout << ' ' << *it;
```

* ++ --

در C++ پیمایشگرها دارای اشاره‌گر ها نیز می‌آرند.

مثالی از پیمایش رو به عقب

```
std::list<int> mylist;
std::list<int>::reverse_iterator      rit;
for (int i=1; i<=5; ++i)
        mylist.push_back(i);
for (rit=mylist.rbegin(); rit!=mylist.rend(); --rit)
        std::cout << ' ' << *rit;
```