Iterators in C++

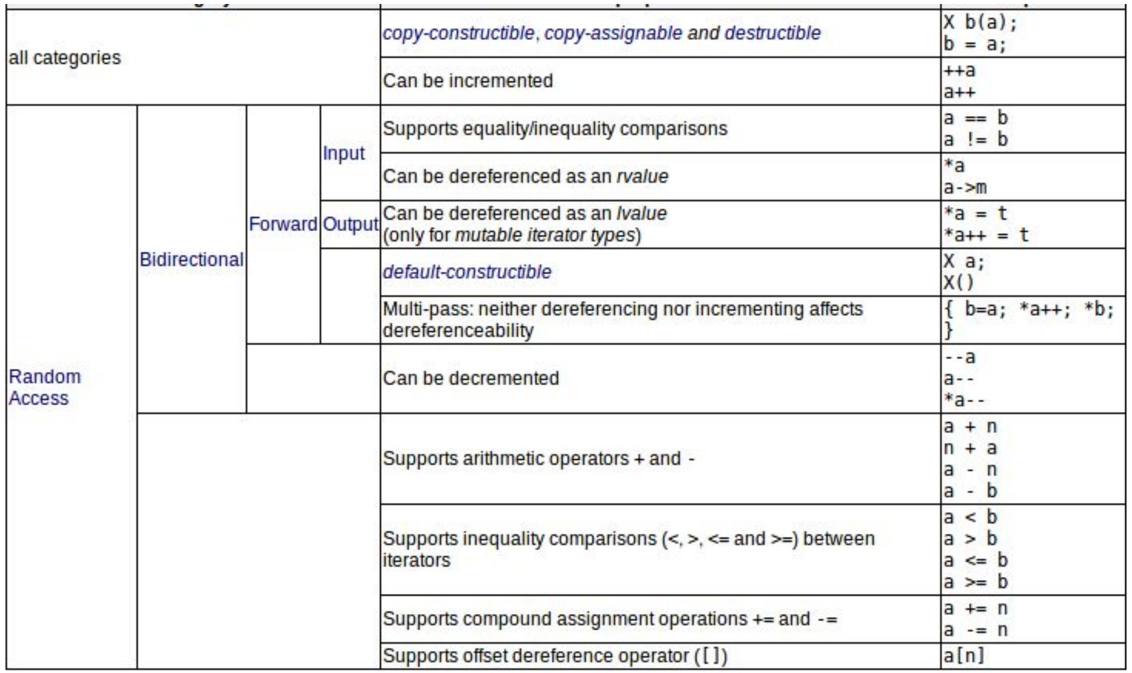
Marcel Vilalta
marcel.vilalta@gmail.com

Base of Iterating and STL

- An iterator is an object that enables a programmer to traverse a container
- For the final user, using iterator must be transparent from its container
- [`begin`, `end`) -----> `begin` == `end`
- Iterators as a pointer to an element
- Different types of iterator within overloading operators

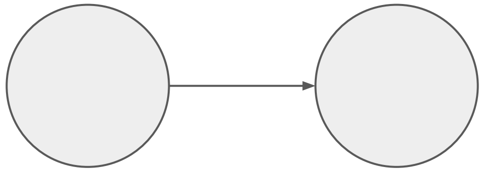
Categories (1)

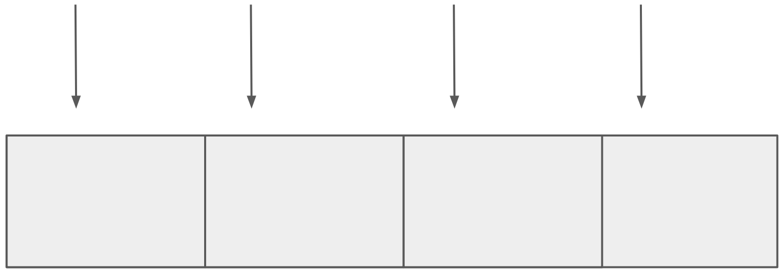
- Input/Output Iterator
- Forward Iterator
- Bidirectional Iterator
- Random-access Iterator



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]`

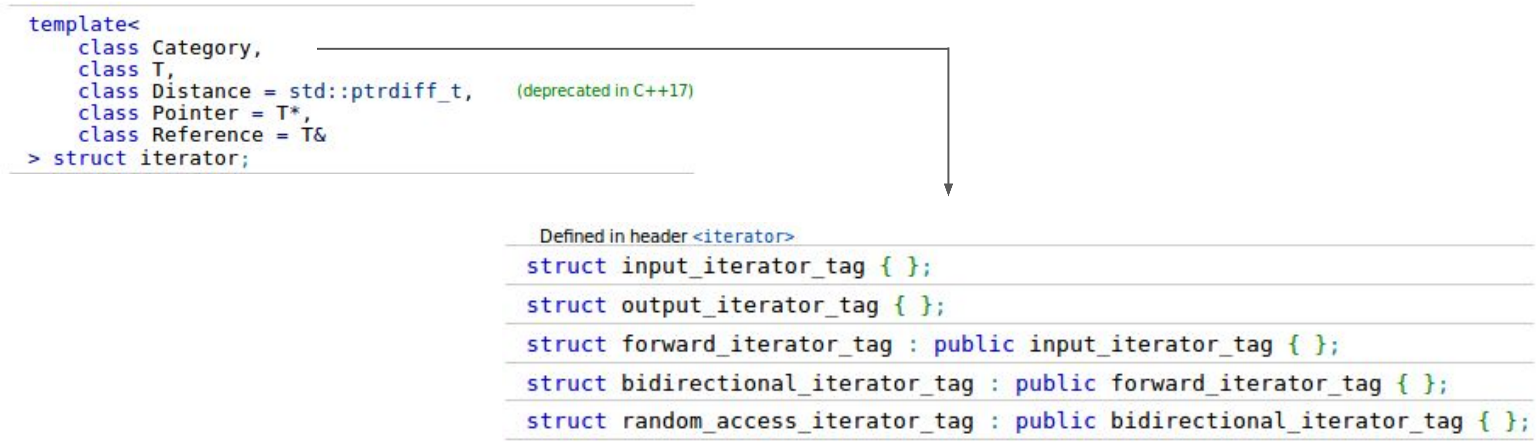
Categories (2)





Iterator traits

- Provides an interfaces to the properties of an iterator.
- *The reason that STL containers and algorithms work so well together, is that they know nothing of each other* - Alex Stepanov



```
template<
    class Category,
    class T,
    class Distance = std::ptrdiff_t,
    class Pointer = T*,
    class Reference = T&
> struct iterator;
```

(deprecated in C++17)

Defined in header `<iterator>`

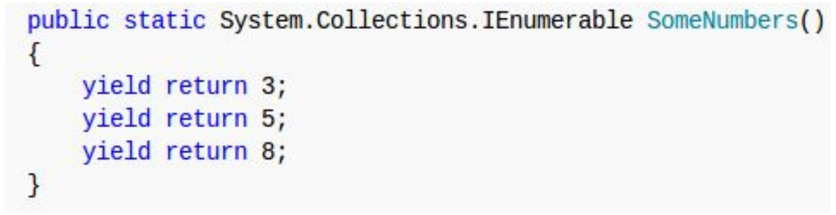
```
struct input_iterator_tag { };
struct output_iterator_tag { };
struct forward_iterator_tag : public input_iterator_tag { };
struct bidirectional_iterator_tag : public forward_iterator_tag { };
struct random_access_iterator_tag : public bidirectional_iterator_tag { };
```

Evolution of <iterator> in C++

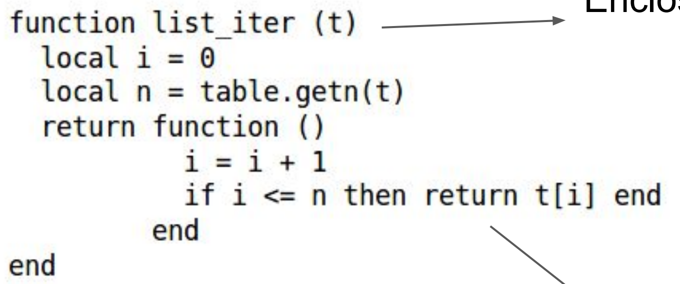
- C++98: Iterator traits defined as **typedef typename**
- C++11: Iterator traits defined as **using**
- **C++17: Deprecating the need inheritance of std::iterator<Category, Distance...> for defining traits**
  - https://www.fluentcpp.com/2018/05/08/std-iterator-deprecated/

Python & C# & Lua & ...

- Iterable
- Generator
- yield concept
- Lua: Clousure concept



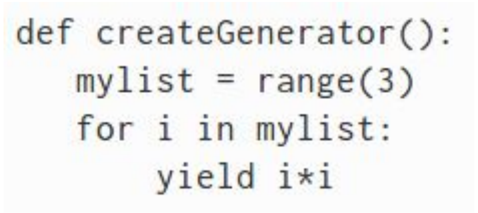
```
public static System.Collections.IEnumerable SomeNumbers()
{
    yield return 3;
    yield return 5;
    yield return 8;
}
```



```
function list_iter (t)
  local i = 0
  local n = table.getn(t)
  return function ()
           i = i + 1
           if i <= n then return t[i] end
         end
end
```

Enclosing function

Closing function



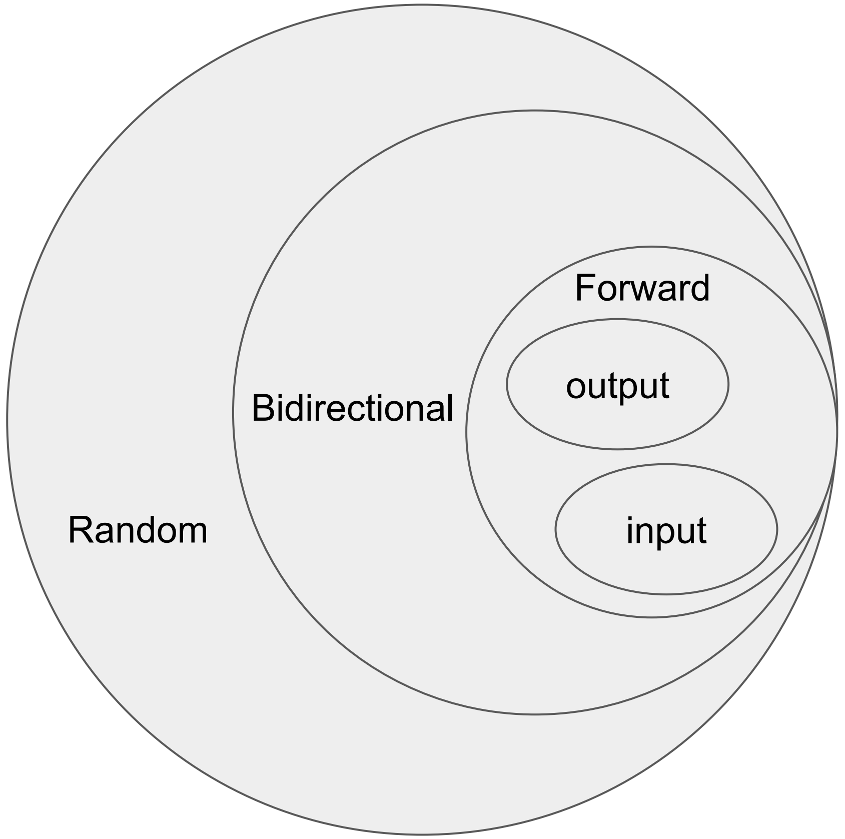
```
def createGenerator():
   mylist = range(3)
   for i in mylist:
       yield i*i
```

Co-routines and Iterators are friends

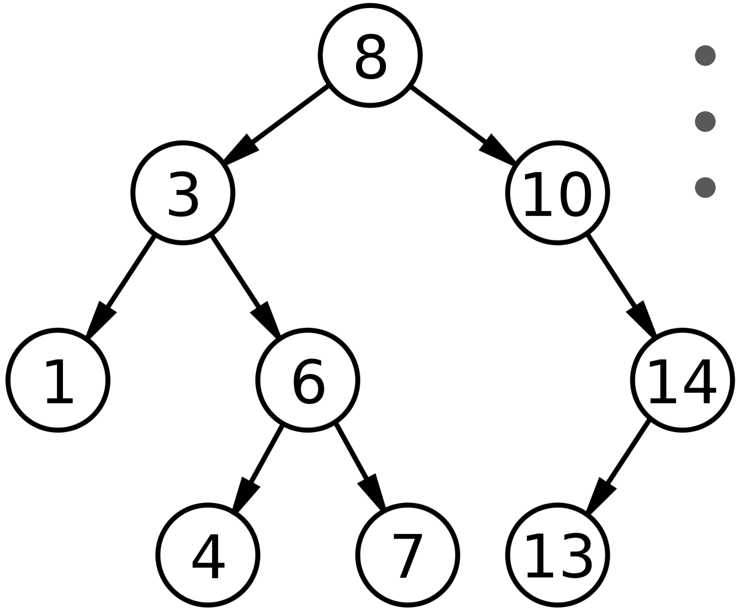
- Remembering Mikel presentation…

- co_yield: yield keyword, returning generator
- co_return: just return a value
- co_await: suspend the coroutine to wait for a thing
- Support for GCC experimental (c++2a)

Custom Data Structure + Custom Iterator = COOL


Forward
output
Bidirectional
input
Random

Binary Search Tree



- PREORDER: Data, Left, Right
- INORDER: Left, Data, Right
- POSTORDER: Left, Right, Data

How many different containers?

How many different iterators?

Is possible to use the same iterator
for a different same-style container?

Custom Iterators for BST

www.github.com/warc3l/cppmeetup/iterator