ITERATOR PATTERN

Group 10

Nguyễn Tăng Bảo Ân

Nguyễn Trịnh Nhật Quang

Bùi Nguyễn Đức Toàn

Definition

- An iterator is any object that, pointing to some element in a range of elements, also called a container. Its intent is to traverse the container, separating the traversal algorithms from the container itself.
- Container example: dynamic arrays, vector, queue, stack, heaps (priority_queue), linked list, trees, associative arrays (map)…

Categories

Random Access Iterator

Read
Increment (w/o multiple passes)
Increment (w/ multiple passes)
Decrement
Random access
Contiguous storage

Categories

Output Iterator

Write
Increment (w/o multiple passes)

<table>
<tr><th colspan="5">Iterator category</th><th>Defined operations</th></tr>
<tr><td rowspan="5">Contiguous Iterator</td><td rowspan="4">Random Access Iterator</td><td rowspan="3">Bidirectional Iterator</td><td rowspan="2">Forward Iterator</td><td>Input Iterator</td><td>▪ read
▪ increment (without multiple passes)</td></tr>
<tr><td></td><td>▪ increment (with multiple passes)</td></tr>
<tr><td colspan="2"></td><td>▪ decrement</td></tr>
<tr><td colspan="3"></td><td>▪ random access</td></tr>
<tr><td colspan="4"></td><td>▪ contiguous storage</td></tr>
<tr><td colspan="6">Iterators that fall into one of the above categories and also meet the requirements of Output Iterator are called mutable iterators.</td></tr>
<tr><td>Output Iterator</td><td colspan="5">▪ write
▪ increment (without multiple passes)</td></tr>
</table>

Design pattern

Some container already have an iterator prepared.

```
void main() {
    vector<string> myvector;

    myvector.push_back("a");
    myvector.push_back("b");
    myvector.push_back("c");
    myvector.push_back("d");

    vector<string>::iterator it;
    int n = 3;
    int i = 0;


    for(it=myvector.begin() ; it < myvector.end(); it++,i++ ) {
        if(i == n) {
            cout<< *it;
            break;
        }
    }

    cout<<myvector[n]<<endl;
    cout<<myvector.at(n)<<endl;
}
```

Problem

Bookshelf simulator

Book **Class**
Fields: - author : string - ISBN : string - title : string **Methods:** + Book() + Book(string title, string author, string ISBN)

Bookshelf **Class**
Fields: - books : Book** - size : size_t **Methods:** + ~Bookshelf() + Bookshelf() + Bookshelf(size_t size) + operator[](size_t i) : Book*&

```cpp
#pragma once
#include <string>
using std::string;

class Book {
public:
    Book();
    Book(string title, string author, string ISBN);
private:
    string title;
    string author;
    string ISBN;
};
```

```cpp
#pragma once
#include "Book.h"

class Bookshelf {
public:
    Bookshelf();
    Bookshelf(size_t size);
    ~Bookshelf();
    //indexing
    Book*& operator[](size_t i);
private:
    Book** books;
    size_t size;
};
```

```cpp
#include "Bookshelf.h"
#include <iostream>
using std::cout;
using std::cin;

int main() {
    Bookshelf myShelf(10);

    myShelf[1] = new Book();
    myShelf[3] = new Book();
    myShelf[6] = new Book();

    return 0;
}
```

Problem

What is the meaning of **next()**?

Problem

Bookshelf simulator **with Iterator**

Bookshelf
Class

Fields:
- books : Book**
- size : size_t

Methods:
+ ~Bookshelf()
+ Bookshelf()
+ Bookshelf(size_t size)
+ operator[](size_t i) : Book*&
+ getIterator(): BookshelfIterator
+ begin() : BookshelfIterator
+ end() : BookshelfIterator

BookshelfIterator
Class

Fields:
- index : size_t
- shelf : const Bookshelf*

Methods:
+ BookshelfIterator(const Bookshelf* shelf)
+ end() : void
+ first() : void
+ operator!= : bool
+ operator*() : Book*&
+ operator++(): BookshelfIterator&
+ operator== : bool

```
class BookshelfIterator {
public:
    BookshelfIterator(const Bookshelf *shelf) {
        this->shelf = shelf;
        index = 0;
    }
    //the position of the first book (if no book then first = end)
    void first() {
        for (index = 0; index < shelf->size; ++index) {
            if (shelf->books[index] != NULL)
                break;
        }
    }
    //the end mark
    void end() {
        index = shelf->size;
    }
    //increment the iterator (the next book)
    BookshelfIterator& operator++() {
        for (++index; index < shelf->size; ++index) {
            if (shelf->books[index] != NULL)
                break;
        }
        return *this;
    }
```

```
    //get the current book of the iterator
    Book*& operator*() {
        return shelf->books[index];
    }
    //compare 2 iterator
    bool operator!= (const BookshelfIterator& iterator) {
        return shelf != iterator.shelf ||
            index != iterator.index;
    }
    bool operator== (const BookshelfIterator& iterator) {
        return shelf == iterator.shelf &&
            index == iterator.index;
    }
private:
    const Bookshelf *shelf;
    size_t index;
};
```

```cpp
#pragma once
#include "Book.h"

class Bookshelf {
public:
    friend class BookshelfIterator;
    Bookshelf();
    Bookshelf(size_t size);
    ~Bookshelf();
    //indexing
    Book*& operator[](size_t i);

    //create a new Iterator
    BookshelfIterator getIterator() const;
    //the begin Iterator
```

```
BookshelfIterator Bookshelf::getIterator() const {
    return BookshelfIterator(this);
}
//the begin Iterator
BookshelfIterator Bookshelf::begin() const{
    BookshelfIterator it(this);
    it.first();
    return it;
}
//the end Iterator
BookshelfIterator Bookshelf::end() const{
    BookshelfIterator it(this);
    it.end();
    return it;
}
```

```cpp
#include "Bookshelf.h"
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

int main() {
    Bookshelf myShelf(10);
    BookshelfIterator it = myShelf.getIterator();

    myShelf[1] = new Book("A", "AA", "1234");
    myShelf[3] = new Book("B", "BB", "3456");
    myShelf[6] = new Book("C", "CC", "7890");

    for (it = myShelf.begin(); it != myShelf.end(); ++it) {
        cout << (*it)->title << endl;
    }
}
```

Problem

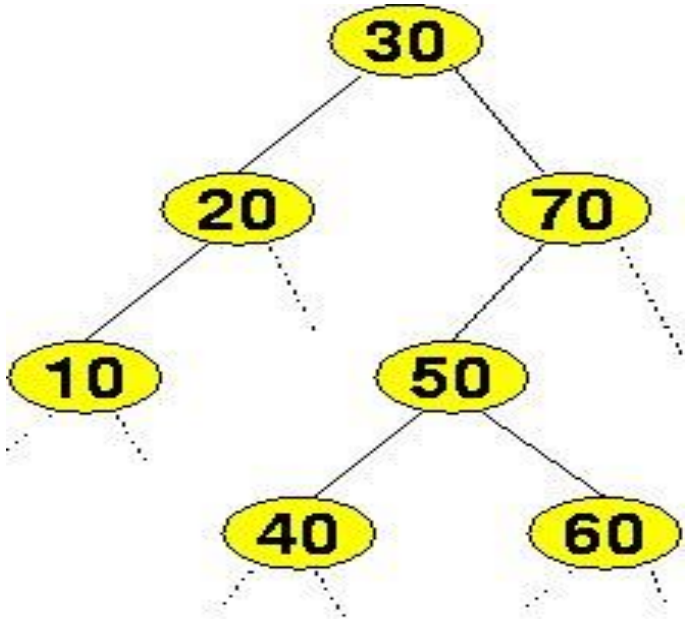
Traversing Trees with Iterators

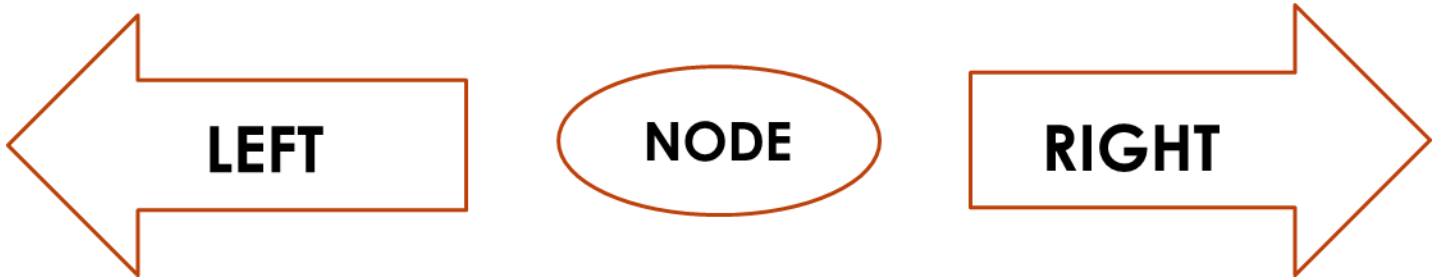
- Begin()
- End()
- Operator++

- The begin will be the leftmost node.

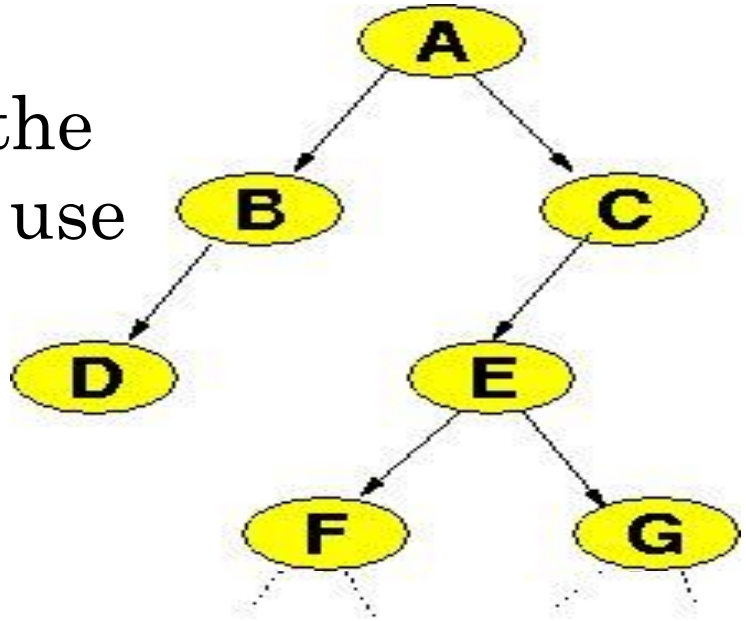
- The end will be the rightmost node.

- We traverse the tree in ascending order.





If our iterator is at the node A, then the iterator will point to what node if we use operator++ ?



- **The iterator give us a uniform way to traverse the binary tree just like the array or linked-list.**
- **Easy to read and understand the code.**

Notes:

- We have 2 kind of iterator “const_iterator” and “iterator”.

- Sequence container traverse.

- Be careful when delete an element.

Advantages

- Provide a uniform interface for traversing different collection and support polymorphic traversal.
- Access contents of a collection without exposing its internal structure.
- Iterators allow you to separate algorithms from the container.
- Many Iterators at once

```
class MyIterator {
    virtual void first() = 0;
    virtual void end() = 0;
    virtual void next() = 0;
    virtual Book*& getItem() = 0;
    ...
};

class TraditionalBookshelfIter : public MyIterator {
    ...
};

class ConventionalBookshelfIter : public MyIterator {
    ...
};
```

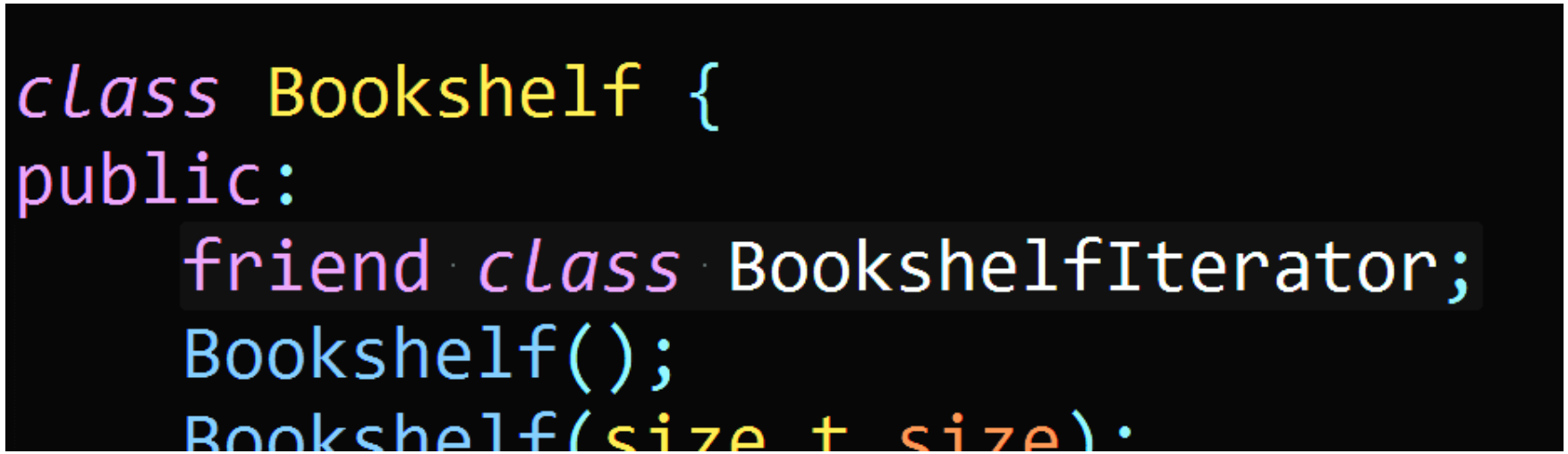
```
class TraditionalBookshelf {
    private Book*;
    TraditionalBookshelfIter* getIterator();
    ...
};

class ConventionalBookshelf {
    private BookLinkedList;
    ConventionalBookshelfIter* getIterator();
    ...
};
```

```
class Librarian {
    TraditionalBookshelf tradShelf;
    ConventionalBookshelf convShelf;
    ...
    void printBook() {
        MyIterator tradIter*;
        MyIterator convIter*;
        tradIter = tradShelf.getIterator();
        convIter = convShelf.getIterator();
        ...
    }
};
```

Disadvantages

- Iterators have access to internal members of the class it aggregates.



Disadvantages

- In multi-threading programming, removing an element stored in an Iterator and accessing it via another Iterator cause undefined behavior.