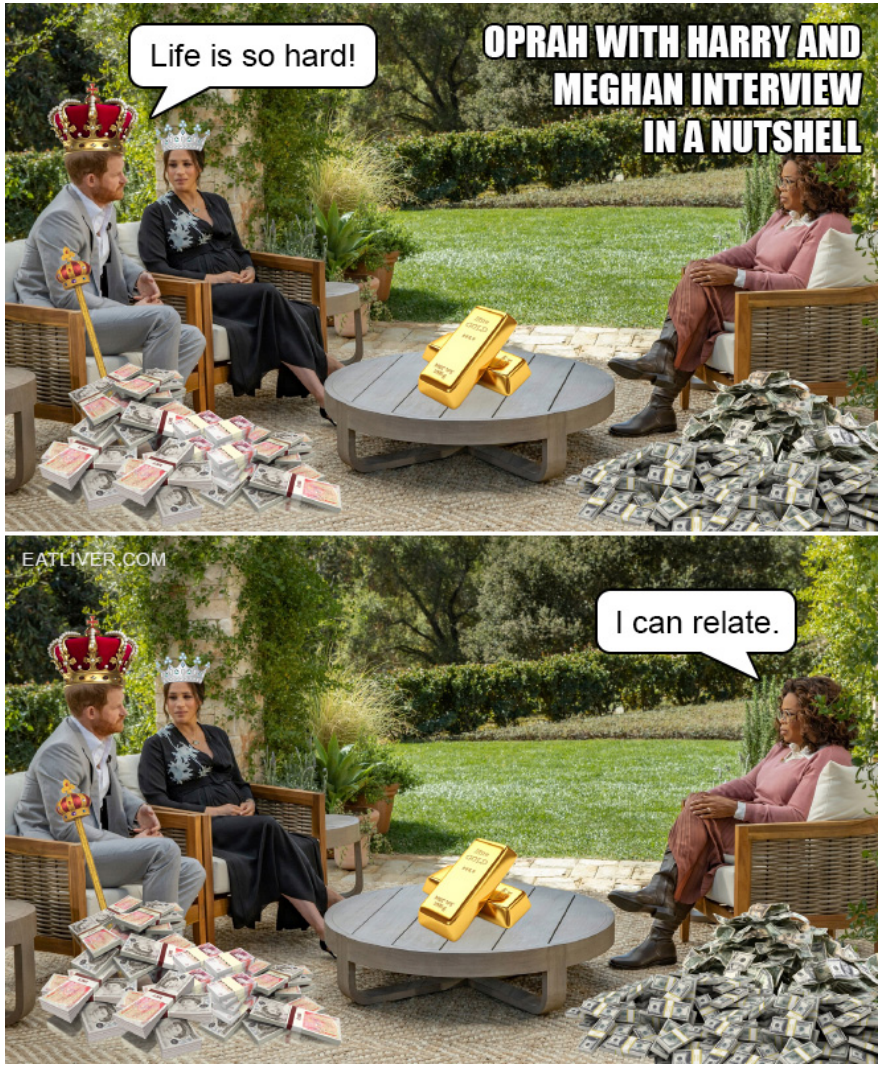
The Standard Template Library

Kafi Rahman

Assistant Professor @CS
Truman State University

Life is hard


Life is so hard!
OPRAH WITH HARRY AND MEGHAN INTERVIEW IN A NUTSHELL
EATLIVER.COM
I can relate.

Mutable Iterators

```
// Define an array object.
array<int, 5> numbers = {1, 2, 3, 4, 5};

// Define an iterator for the array object.
array<int, 5>::iterator it;

// Make the iterator point to the array object's first element.
it = numbers.begin();

// Use the iterator to change the element.
*it = 99;
```

- An iterator gives you read/write access to the element to which the iterator points.
- This is commonly known as a mutable iterator.

Constant Iterators

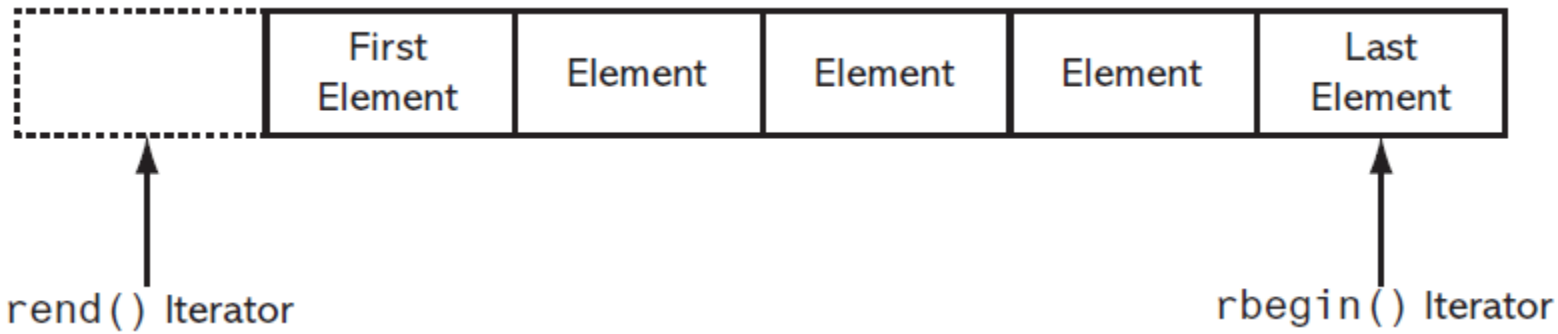
- An iterator of the const_iterator type provides read-only access to the element to which the iterator points.

- The STL containers provide a cbegin() member function and a cend() member function.
  - The cbegin() member function returns a const_iterator pointing to the first element in a container.
  - The cend() member function returns a const_iterator pointing to the end of the container.
  - When working with const_iterators, simply use the container class's cbegin() and cend() member functions instead of the begin() and end() member functions.

Reverse Iterators

- A reverse iterator works in reverse, allowing you to iterate backward over the elements in a container.
- With a reverse iterator, the last element in a container is considered the first element, and the first element is considered the last element.
- The ++ operator moves a reverse iterator backward, and the –– operator moves a reverse iterator forward.

Reverse Iterators



- The rbegin() member function returns a reverse iterator pointing to the last element in a container.
- The rend() member function returns an iterator pointing to the position before the first element.

Reverse Iterators

```
// Define an array object.
array<int, 5> numbers = {1, 2, 3, 4, 5};

// Define a reverse iterator for the array object.
array<int, 5>::reverse_iterator it;

// Display the elements in reverse order.
for (it = numbers.rbegin(); it != numbers.rend(); it++)
   cout << *it << endl;
```

- To create a reverse iterator, define it as reverse_iterator

17.3

The vector Class

Expert birdwatcher

The vector Class

- A vector is a sequence container that works like an array, but is dynamic in size.
- Overloaded [] operator provides access to existing elements
- The vector class is declared in the <vector> header file.

vector Class Constructors

Default Constructor	vector<dataType> name; Creates an empty vector.
Fill Constructor	vector<dataType> name(size); Creates a vector of size elements. If the elements are objects, they are initialized via their default constructor. Otherwise, initialized with 0.
Fill Constructor	vector<dataType> name(size, value); Creates a vector of size elements, each initialized with value.

vector Class Constructors

Range Constructor	vector<dataType> name(iterator1, iterator2); Creates a vector that is initialized with a range of values from another container. iterator1 marks the beginning of the range and iterator2 marks the end.
Copy Constructor	vector<dataType> name(vector2); Creates a vector that is a copy of vector2.

vector Class Constructors

Program 17-4

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    const int SIZE = 10;

    // Define a vector to hold 10 int values.
    vector<int> numbers(SIZE);

    // Store the values 0 through 9 in the vector.
    for (int index = 0; index < numbers.size(); index++)
       numbers[index] = index;

    // Display the vector elements.
    for (auto element : numbers)
       cout << element << " ";
    cout << endl;

    return 0;
}
```

Subscript notation

supports for each loop

Program Output

```
0 1 2 3 4 5 6 7 8 9
```

Initializing a vector

- In C++ 11 and later, you can initialize a vector object:

```
vector<int> numbers = {1, 2, 3, 4, 5};

// or

vector<int> numbers {1, 2, 3, 4, 5};
```

Adding New Elements to a vector

- The push_back member function adds a new element to the end of a vector:

```
vector<int> numbers;
numbers.push_back(10);
numbers.push_back(20);
numbers.push_back(30);
```

Advanced student

Accessing Elements with the at() Member Function

- You can use the at() member function to retrieve a vector element by its index with bounds checking:

```
vector<string> names = {"Joe", "Karen", "Lisa"};
cout << names.at(0) << endl;
cout << names.at(1) << endl;
cout << names.at(2) << endl;

// Throws an exception
cout << names.at(3) << endl;
```

Throws an out_of_bounds exception when given an invalid index

Using an Iterator With a vector

```
// Create a vector containing names.
vector<string> names = {"Joe", "Karen", "Lisa", "Jackie"};

// Create an iterator.
vector<string>::iterator it;

// Use the iterator to display each element in the vector.
for (it = names.begin(); it != names.end(); it++)
{
   cout << *it << endl;
}
```

Defines an iterator that is compatible with a vector<string> object

Displays the item that the iterator points to

- vectors have begin() and end() member functions that return iterators pointing to the beginning and end of the container:

Using an Iterator With a vector

- The begin() and end() member functions return a random-access iterator of the iterator type
- The cbegin() and cend() member functions return a random-access iterator of the const_iterator type
- The rbegin() and rend() member functions return a reverse iterator of the reverse_iterator type
- The crbegin() and crend() member functions return a reverse iterator of the const_reverse_iterator type

Inserting Elements with the insert() Member Function

- You can use the insert() member function, along with an iterator, to insert an element at a specific position.
- General format:

```
vectorName.insert(it, value);
```

Iterator pointing to an element in the vector

Value to insert before the element that it points to

Inserting Elements with the insert() Member Function

Program 17-5

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Define a vector with 5 int values.
    vector<int> numbers = {1, 2, 3, 4, 5};

    // Define an iterator pointing to the second element.
    auto it = numbers.begin() + 1;

    // Insert a new element with the value 99.
    numbers.insert(it, 99);

    // Display the vector elements.
    for (auto element : numbers)
       cout << element << " ";
    cout << endl;

    return 0;
}
```

Program Output

```
1 99 2 3 4 5
```

Overloaded Versions of the insert() Member Function

insert(it, value)	— Inserts value just before the element pointed to by it. — The function returns an iterator pointing to the newly inserted element.
insert(iterator1, iterator2, iterator3)	— Inserts a range of new elements. iterator1 points to an existing element in the container. The range of new elements will be inserted before the element pointed to by iterator1. — Here, iterator2 and iterator3 mark the beginning and end of a range of values that will be inserted. (The element pointed to by iterator3 will not be included in the range.) — The function returns an iterator pointing to the first element of the newly inserted range.

Storing Objects Of Your Own Classes in a vector

- STL containers are especially useful for storing objects of your own classes.
- Consider this Product class:

```
#ifndef PRODUCT_H
#define PRODUCT_H
#include <string>
using namespace std;

class Product
{
private:
    string name;
    int units;
public:
    Product(string n, int u)
    {   name = n;
        units = u; }

    void setName(string n)
    {   name = n; }

    void setUnits(int u)
    {   units = u; }

    string getName() const
    {   return name; }

    int getUnits() const
    {   return units; }
};
#endif
```

Storing Objects Of Your Own Classes in a vector

Program 17-7

```
#include <iostream>
#include <vector>
#include "Product.h"
using namespace std;

int main()
{
    // Create a vector of Product objects.
    vector<Product> products =
    {
        Product("T-Shirt", 20),
        Product("Calendar", 25),
        Product("Coffee Mug", 30)
    };

    // Display the vector elements.
    for (auto element : products)
    {
        cout << "Product: " << element.getName() << endl
             << "Units: " << element.getUnits() << endl;
    }

    return 0;
}
```

This program initializes a vector with three Product objects.

A range-based for loop iterates over the vector.

Program Output

```
Product: T-Shirt
Units: 20
Product: Calendar
Units: 25
Product: Coffee Mug
Units: 30
```

Storing Objects Of Your Own Classes in a vector

Program 17-8

```
#include <iostream>
#include <string>
#include <vector>
#include "Product.h"
using namespace std;

int main()
{
    // Create Product objects.
    Product prod1("T-Shirt", 20);
    Product prod2("Calendar", 25);
    Product prod3("Coffee Mug", 30);

    // Create a vector to hold the Products
    vector<Product> products;

    // Add the products to the vector.
    products.push_back(prod1);
    products.push_back(prod2);
    products.push_back(prod3);

    // Use an iterator to display the vector contents.
    for (auto it = products.begin(); it != products.end(); it++)
    {
        cout << "Product: " << it->getName() << endl
             << "Units: " << it->getUnits() << endl;
    }

    return 0;
}
```

This program uses the push_back member function to store three Product objects in a vector.

A for loop uses an iterator to step through the vector.

Program Output

```
Product: T-Shirt
Units: 20
Product: Calendar
Units: 25
Product: Coffee Mug
Units: 30
```

Inserting Container Elements With Emplacement

- Member functions such as insert() and push_back() can cause temporary objects to be created in memory while the insertion is taking place.
- This is not a problem in programs that make only a few insertions.
- However, these functions can be inefficient for making a lot of insertions.

Inserting Container Elements With Emplacement

- C++11 introduced a new family of member functions that use a technique known as emplacement to insert new elements.
- Emplacement avoids the creation of temporary objects in memory while a new object is being inserted into a container.
- The emplacement functions are more efficient than functions such as insert() and push_back()

Inserting Container Elements With Emplacement

- The vector class provides two member functions that use emplacement:
  - `emplace()` - emplaces an element at a specific location
  - `emplace_back()`- emplaces an element at the end of the vector
- With these member functions, it is not necessary to instantiate, ahead of time, the object you are going to insert.
- Instead, you pass to the emplacement function any arguments that you would normally pass to the constructor of the object you are inserting.
- The emplacement function handles the construction of the object, forwarding the arguments to its constructor.

Inserting Container Elements With Emplacement

Program 17-9

```
#include <iostream>
#include <vector>
#include "Product.h"
using namespace std;

int main()
{
    // Create a vector to hold Products.
    vector<Product> products;

    // Add Products to the vector.
    products.emplace_back("T-Shirt", 20);
    products.emplace_back("Calendar", 25);
    products.emplace_back("Coffee Mug", 30);

    // Use an iterator to display the vector contents.
    for (auto it = products.begin(); it != products.end(); it++)
    {
        cout << "Product: " << it->getName() << endl
             << "Units: " << it->getUnits() << endl;
    }

    return 0;
}
```

Define a vector to hold Product objects

Emplace three Product objects at the end of the vector

A for loop uses an iterator to step through the vector.

Program Output

```
Product: T-Shirt
Units: 20
Product: Calendar
Units: 25
Product: Coffee Mug
Units: 30
```

Inserting Container Elements With Emplacement

Program 17-10

```
#include <iostream>
#include <vector>
#include "Product.h"
using namespace std;

int main()
{
    // Create a vector to hold Products.
    vector<Product> products =
    {
        Product("T-Shirt", 20),
        Product("Coffee Mug", 30)
    };

    // Get an iterator to the 2nd element.
    auto it = products.begin() + 1;

    // Insert another Product into the vector.
    products.emplace(it, "Calendar", 25);

    // Display the vector contents.
    for (auto element : products)
    {
        cout << "Product: " << element.getName() << endl
             << "Units: " << element.getUnits() << endl;
    }

    return 0;
}
```

Program Output

```
Product: T-Shirt
Units: 20
Product: Calendar
Units: 25
Product: Coffee Mug
Units: 30
```

Initializes a vector with two Product objects

Gets an iterator pointing to the 2nd element

Emplaces a new Product object before the one pointed to by the iterator

The vector Class

- The vector class has many useful member functions.
- See Table 17-8 in your textbook.