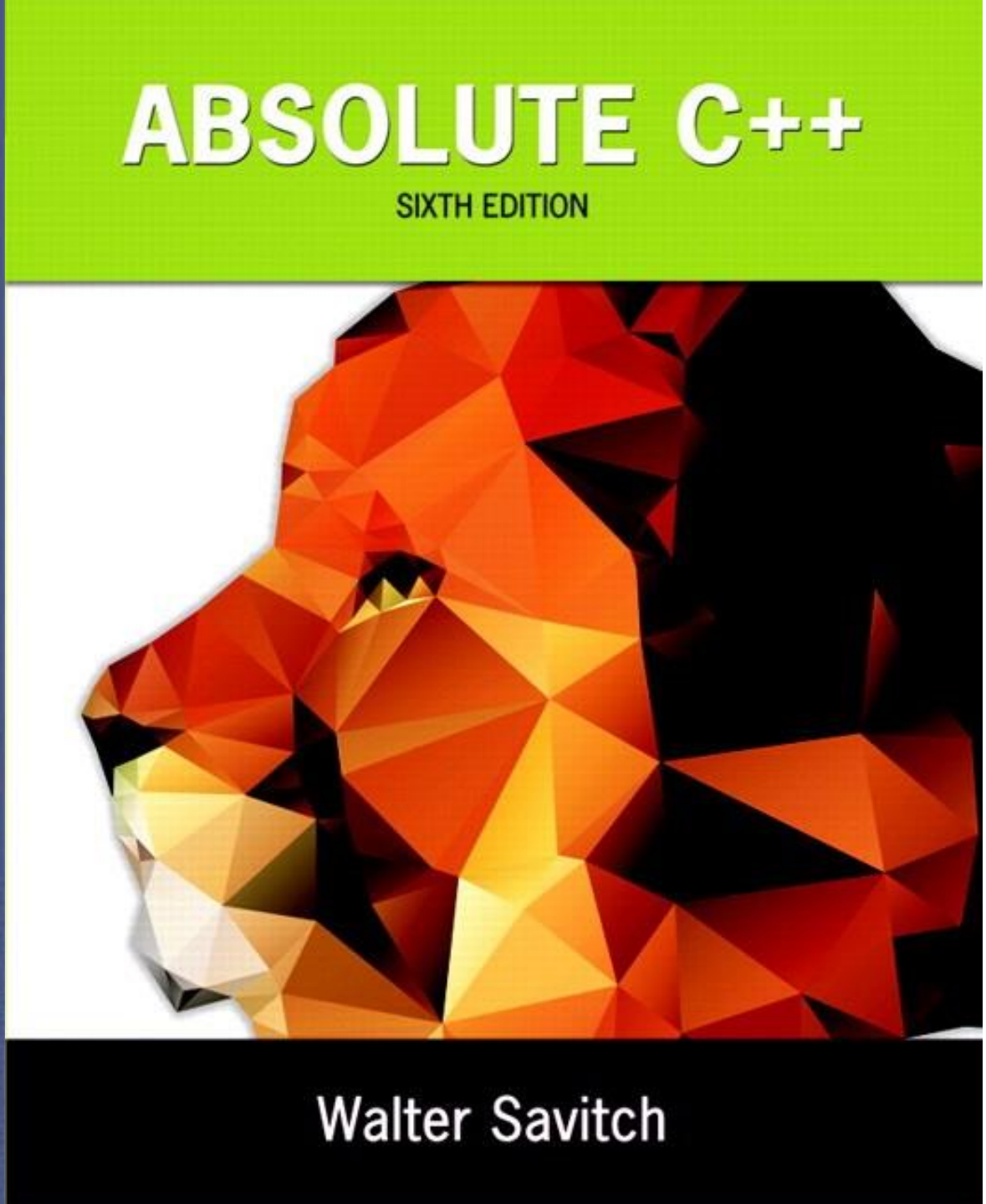


Chapter 19

Standard Template Library

PEARSON

Introduction

- Standard Template Library (STL)
  - 스택과 큐와 같은 표준 데이터 구조들을 위한 라이브러리들을 포함하고 있다.

Iterators

- 포인터 개념을 객체 관점에서 추상화한 것이다.
  - 구현 디테일을 숨기기 위해 설계됨
  - 다른 컨테이너 클래스들에 대해서 하나의 공통적인 인터페이스를 제공함
- 각 컨테이너 클래스는 자기 자신만의 iterator 타입을 가지고 있다.
  - 각 데이터 타입에 자기 자신만의 포인터 타입을 갖는 것과 유사하다.

Manipulating Iterators

- 오버로드된 연산자들을 사용할 수 있다:
  - ++, --, ==, !=
  - *
    - p 가 iterator 객체이면, *p 는 p 가 가리키는 데이터를 접근하는 것이다.
- Vector template class
  - 위에서 열거한 연산자들을 모두 가지고 있다.
  - 또한, begin() 과 end() 라는 멤버 함수들을 가지고 있다.
    c.begin(); // c 내부 1st 아이템에 대한 iterator 리턴
    c.end(); // c 내부 아이템들의 끝을 의미하는 test 값 리턴

Cycling with Iterators

- iterator 를 이용한 모든 아이템 방문하기:

```
for (p = c.begin(); p != c.end(); p++)
        process *p
```

- STL 의 각 컨테이너 타입은 자신만의 iterator 타입을 가지고 있다.
  - 그러나 사용법은 모두 유사하다.

Display 19.1
Iterators Used with a Vector (1 of 2)

```
1       //Program to demonstrate STL iterators.
2       #include <iostream>
3       #include <vector>
4       using std::cout;
5       using std::endl;
6       using std::vector;

7       int main( )
8       {
9           vector<int> container;

10          for (int i = 1; i <= 4; i++)
11              container.push_back(i);

12          cout << "Here is what is in the container:\n";
13          vector<int>::iterator p;
14          for (p = container.begin( ); p != container.end( ); p++)
15              cout << *p << " ";
16          cout << endl;

17          cout << "Setting entries to 0:\n";
18          for (p = container.begin( ); p != container.end( ); p++)
19               *p = 0;
```

Display 19.1
Iterators Used with a Vector (2 of 2)

```
20          cout << "Container now contains:\n";
21          for (p = container.begin( ); p !=
                    container.end( ); p++)
22              cout << *p << " ";
23          cout << endl;

24          return 0;
25      }
```

SAMPLE DIALOGUE

Here is what is in the container:
1 2 3 4
Setting entries to 0:
Container now contains:
0 0 0 0

Vector Iterator Types

- vector<int> 의 iterator 타입

  std::vector<int>::iterator

- list<int> 의 iterator 타입

  std::list<int>::iterator

Random Access:
Display 19.2 Bidirectional and Random-Access Iterator Use

```
 7  int main( )
 8  {
 9      vector<char> container;

10      container.push_back('A');
11      container.push_back('B');
12      container.push_back('C');
13      container.push_back('D');

14      for (int i = 0; i < 4; i++)
15          cout << "container[" << i << "] == "
16               << container[i] << endl;

17      vector<char>::iterator p = container.begin( );
18      cout << "The third entry is " << container[2] << endl;
19      cout << "The third entry is " << p[2] << endl;
20      cout << "The third entry is " << *(p + 2) << endl;

21      cout << "Back to container[0].\n";
22      p = container.begin( );
23      cout << "which has value " << *p << endl;

24      cout << "Two steps forward and one step back:\n";
25      p++;
26      cout << *p << endl;
```

Three different notations for the same thing

This notation is specialized to vectors and arrays.

These two work for any random-access iterator.

Iterator Classifications

- Forward iterators:
  - 연산자 ++ 만 동작한다.
- Bidirectional iterators:
  - 연산자 ++ 와 -- 도 동작한다.
- Random-access iterators:
  - 연산자 ++ 와 – 뿐만 아니라, random access 도 동작한다.

Constant and Mutable Iterators

- Dereferencing 연산자의 동작을 규정한다.
- Constant iterator:
  - * 연산자가 read-only version of element 를 생산한다.
  - *p 표현을 통해서 컨테이너 내부 원소의 내용을 읽을 수는 있지만, 원소 내용을 변경할 수는 없다.
    - 즉, *p 표현은 right-value 로만 사용될 수 있다.
- Mutable iterator:
  - *p 표현을 통해서 컨테이너 내부 원소의 내용을 변경할 수 있다.
  - 즉, *p 표현은 left-value 와 right-value 로 모두 사용될 수 있다.

Reverse Iterators

- 컨테이너 내부 원소들을 역순으로 방문할 때 사용한다.
  - 컨테이너가 bidirectional iterator 를 제공해야 한다.
- 다음 코드 조각을 상상할 수도 있으나, 대부분의 시스템에서는 동작하지 않는다:

```
iterator p;
for (p=container.end();p!=container.begin(); p--)
        cout << *p << " " ;
```

  - end() 는 sentinel 값만 리턴한다는 것을 기억하라.

Reverse Iterators Correct

- 올바른 코드 예시는 다음과 같다:

```
reverse_iterator p;
for (rp=container.rbegin();rp!=container.rend(); rp++)
        cout << *rp << " " ;
```

- rbegin()
  - 마지막 원소를 위한 iterator 를 리턴한다.
- rend()
  - sentinel " end " 값을 리턴한다.

Compiler Problems

- 어떤 컴파일러들은 iterator 선언에 문제를 일으키기도 한다.
- 예시:

```
using std::vector<char>::iterator;
…
iterator p;
```

- 위 예시대로 컴파일되지 않으면, 다음과 같이 선언하라:

```
std::vector<char>::iterator p;
```

Auto

- C++11 문법의 **auto** 키워드는 템플릿과 iterator 들과 함께 사용되어, 코드의 가독성을 높여준다.
- 예시:

```
vector<int>::iterator p = v.begin();
```

→ `auto p = v.begin();`

Containers

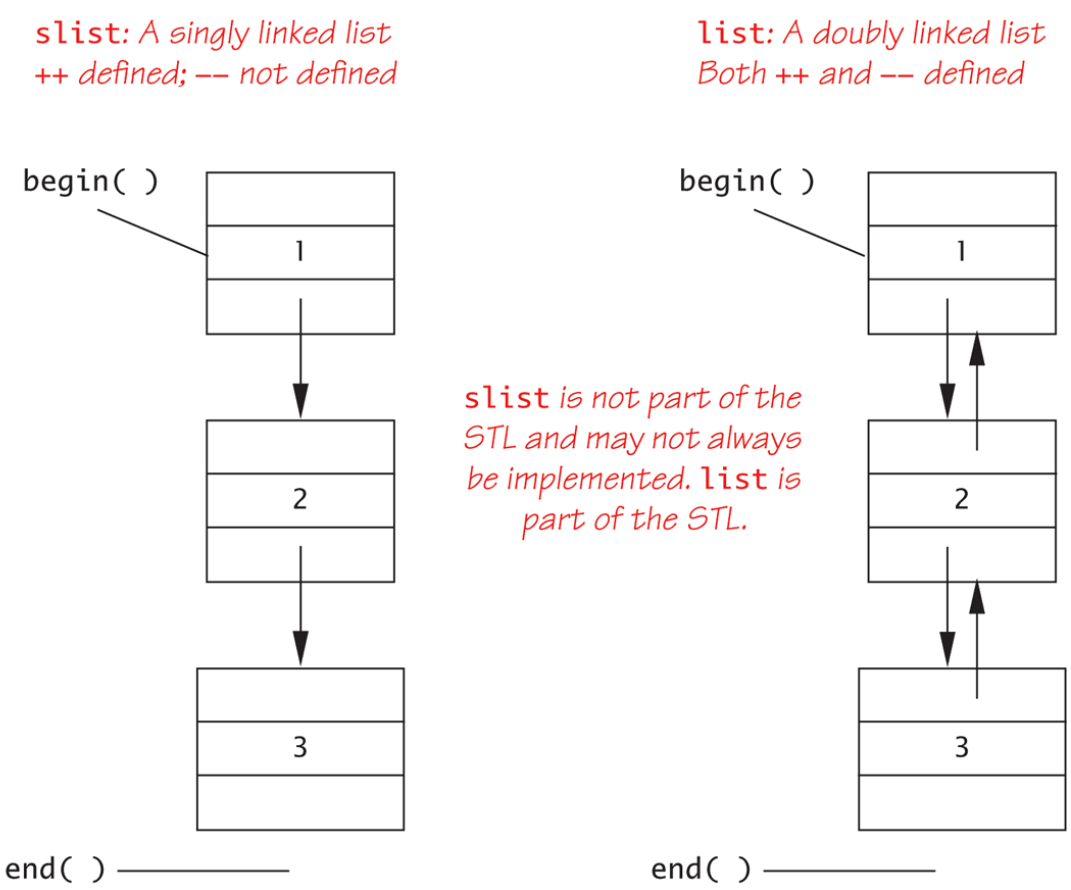
- STL 에 포함된 컨테이너 클래스들
  - 리스트, 큐, 스택 등
  - 각 컨테이너는 템플릿 클래스이다.
  - 각 컨테이너는 자신만의 iterator 를 갖는다.
    - 어떤 컨테이너는 bidirectional iterator, 또 다른 컨테이너는 forward iterator 를 가질 수도 있다.
  - 그러나, 모든 연산자들과 멤버들은 동일한 의미를 갖는다.

Sequential Containers

- 원소들을 순서대로 저장하는 컨테이너
  - 예: Linked list
- STL 의 list 클래스는 “doubly linked list” 로 구현된다.

Display 19.4 Two Kinds of Lists

Display 19.4 Two Kinds of Lists



Display 19.5
Using the list Template Class(1 of 2)

```
1       //Program to demonstrate the STL template class list.
2       #include <iostream>
3       #include <list>
4       using std::cout;
5       using std::endl;
6       using std::list;

7       int main( )
8       {
9           list<int> listObject;

10          for (int i = 1; i <= 3; i++)
11              listObject.push_back(i);

12          cout << "List contains:\n";
13          list<int>::iterator iter;
14          for (iter = listObject.begin( ); iter != listObject.end( );
                 iter++)
15              cout << *iter << " ";
16          cout << endl;
```

Display 19.5
Using the list Template Class(2 of 2)

```
17          cout << "Setting all entries to 0:\n";
18          for (iter = listObject.begin( ); iter != listObject.end( );
                        iter++)
19              *iter = 0;

20          cout << "List now contains:\n";
21          for (iter = listObject.begin( ); iter != listObject.end( );
                        iter++)
22              cout << *iter << " ";
23          cout << endl;

24          return 0;
25      }
```

SAMPLE DIALOGUE

```
List contains:
1 2 3
Setting all entries to 0:
List now contains:
0 0 0
```

Container Adapters

- 컨테이너 어댑터들은 템플릿 클래스이다.
  - 다른 컨테이너 클래스 위에서 구현되었다.
- 어댑터 클래스들은 밑에 디폴트 컨테이너들을 가지고 있다.
  - 예시:
    stack → deque
    priority_queue → vector
- 디폴트 컨테이너를 변경할 수 있다.
  - 예시: stack<int, vector<int> >
    - 디폴트 컨테이너를 벡터로 변경함

Note space between > >

Associative Containers

- 연관 컨테이너는 각 데이터(원소)에 키를 연관시켜 저장한다.
  - 간단한 database 기능 수행
- 예시:
  data: 학생 정보 (연락처, 주소, 수강 과목들...)
  key: 학번

set Template Class

- 가장 단순한 연관 컨테이너
- 원소들을 반복해서 저장하지 않음
- 기본 기능들:
  - Add elements
  - Delete elements
  - Ask if element is in set

More set Template Class

- 효율적으로 설계되었음
  - 원소들을 정렬하여 저장함
  - 정렬 방식을 지정할 수 있음:

    set<T, *Ordering*> s;

    - Ordering 은 bool 값을 리턴하는 잘 정의된 ordering 관계이어야 한다.
    - Ordering 을 비워두면, '<' 관계 연산자가 사용된다.

Program Using the set Template Class (1 of 2)

```
1       //Program to demonstrate use of the set template class.
2       #include <iostream>
3       #include <set>
4       using std::cout;
5       using std::endl;
6       using std::set;

7       int main( )
8       {
9           set<char> s;

10          s.insert('A');
11          s.insert('D');
12          s.insert('D');
13          s.insert('C');
14          s.insert('C');
15          s.insert('B');

16          cout << "The set contains:\n";
17          set<char>::const_iterator p;
18          for (p = s.begin( ); p != s.end( ); p++)
19          cout << *p << " ";
20          cout << endl;
```

Program Using the set Template Class (2 of 2)

```
21        cout << "Set contains 'C': ";
22        if (s.find('C')==s.end( ))
23           cout << " no " << endl;
24        else
26           cout << " yes " << endl;

27           cout << "Removing C.\n";
28           s.erase('C');
29           for (p = s.begin( ); p != s.end( ); p++)
30           cout << *p << " ";
31           cout << endl;

32        cout << "Set contains 'C': ";
33        if (s.find('C')==s.end( ))
34            cout << " no " << endl;
35        else
36            cout << " yes " << endl;

37           return 0;
38      }
```

SAMPLE DIALOGUE

```
The set contains:
A B C D
Set contains 'C':  yes
Removing C.
A B D
Set contains 'C':  no
```

Map Template Class

- (key, value)들의 집합
- 예시 선언:

  map<string, int> numberMap;
- [ ] 연산자 사용 가능
  - value 검색과 저장을 위해
- set 클래스처럼 원소들을 key 기준으로 정렬하여 저장함
  - value 는 정렬에 아무 영향을 줄 수 없음

Program Using the map Template Class (1 of 3)

```
1       //Program to demonstrate use of the map template class.
 2      #include <iostream>
 3      #include <map>
 4      #include <string>
 5      using std::cout;
 6      using std::endl;
 7      using std::map;
 8      using std::string;

 9      int main( )
10      {
11          map<string, string> planets;

12          planets["Mercury"] = "Hot planet";
13          planets["Venus"] = "Atmosphere of sulfuric acid";
14          planets["Earth"] = "Home";
15          planets["Mars"] = "The Red Planet";
16          planets["Jupiter"] = "Largest planet in our solar system";
17          planets["Saturn"] = "Has rings";
18          planets["Uranus"] = "Tilts on its side";
19          planets["Neptune"] = "1500 mile per hour winds";
20          planets["Pluto"] = "Dwarf planet";
```

Program Using the map Template Class (2 of 3)

```
21          cout << "Entry for Mercury - " << planets["Mercury"]
22               << endl << endl;

23          if (planets.find("Mercury") != planets.end())
24              cout << "Mercury is in the map." << endl;
25          if (planets.find("Ceres") == planets.end())
26              cout << "Ceres is not in the map." << endl << endl;

27          cout << "Iterating through all planets: " << endl;
28          map<string, string>::const_iterator iter;
29          for (iter = planets.begin(); iter != planets.end(); iter++)
30          {
31              cout << iter->first << " - " << iter->second << endl;
32          }
```

The iterator will output the map in order sorted by the key. In this case the output will be listed alphabetically by planet.

```
33          return 0;
34      }
```

Program Using the map Template Class (3 of 3)

```
SAMPLE DIALOGUE

Entry for Mercury - Hot planet

Mercury is in the map.
Ceres is not in the map.

Iterating through all planets:
Earth - Home
Jupiter - Largest planet in our solar system
Mars - The Red Planet
Mercury - Hot planet
Neptune - 1500 mile per hour winds
Pluto - Dwarf planet
Saturn - Has rings
Uranus - Tilts on its side
Venus - Atmosphere of sulfuric acid
```

컨테이너를 위한 C++11 문법

- C++11 문법으로 컨테이너의 모든 원소 방문을 쉽게 코딩할 수 있다.
- 예시:

```
map<int, string> personIDs = {
                                {1,"Walt"},
                                {2,"Kenrick"}
                              };
set<string> colors = {"red","green","blue"};

for (auto p : personIDs)
       cout << p.first << " " << p.second << endl;
for (auto p : colors)
       cout << p << " ";
```

Container Access Running Times

- O(1) - constant operation always:
  - Vector inserts to front or back
  - deque inserts
  - list inserts
- O(N)
  - Insert or delete of arbitrary element in vector or deque (N is number of elements)
- O(log N)
  - set or map finding

Nonmodifying Sequence Algorithms

- 컨테이너들에 대해서 동작하는 템플릿 함수들
  - 컨테이너 내용물은 변경하지 않음
- 예시: find 함수
  - 아무 STL sequence container 에 대해서 사용 가능함

Display 19.17
The Generic find Function (1 of 3)

```
1       //Program to demonstrate use of the generic find function.
2       #include <iostream>
3       #include <vector>
4       #include <algorithm>
5       using std::cin;
6       using std::cout;
7       using std::endl;
8       using std::vector;
9       using std::find;

10      int main( )
11      {
12          vector<char> line;

13          cout << "Enter a line of text:\n";
14          char next;
15          cin.get(next);
16          while (next != '\n')
17          {
18              line.push_back(next);
19              cin.get(next);
20          }
```

Display 19.17
The Generic find Function (2 of 3)

```
21          vector<char>::const_iterator where;
22          where = find(line.begin( ), line.end( ), 'e');
23          //where is located at the first occurrence of 'e' in v.

24          vector<char>::const_iterator p;
25          cout << "You entered the following before you entered your
                 first e:\n";
26          for (p = line.begin( ); p != where; p++)
27              cout << *p;
28          cout << endl;

29          cout << "You entered the following after that:\n";
30          for (p = where; p != line.end( ); p++)
31              cout << *p;
32          cout << endl;

33          cout << "End of demonstration.\n";
34          return 0;
35  }
```

If find does not find what it is looking for, it returns its second argument.

Display 19.17
The Generic find Function (3 of 3)

SAMPLE DIALOGUE 1

Enter a line of text
A line of text.
You entered the following before you entered your first e:
A lin
You entered the following after that:
e of text.
End of demonstration.

SAMPLE DIALOGUE 2
Enter a line of text
I will not!
You entered the following before you entered your first e:
I will not!
You entered the following after that:

End of demonstration.

Modifying Sequence Algorithms

- 컨테이너 내용물을 변경하는 STL 함수들
- 원소 추가/삭제는 iterator 에 영향을 줄 수 있음을 기억할 것!
  - list 와 slist 는 iterator 변경되지 않음을 보장함
  - vector 와 deque 는 보장하지 않음

Set Algorithms

- STL 의 집합 연산 함수들
- 이 함수들은 컨테이너들이 내부 원소들을 정렬하여 저장한다고 가정함
  - 예시: set, map, multiset, multimap
    - 집합 연산 함수들 적용 가능함
  - 반례: vector
    - 집합 연산 함수들 적용 불가함

Sorting Algorithms

- STL 은 두 개의 템플릿 함수들을 제공한다:
  1. 원소들을 정렬하는 함수
  2. 정렬된 두 컨테이너를 병합하는 함수
- Guaranteed running time O(N log N)
  - 이것보다 더 빠른 정렬 함수는 없다.