



템플릿과 표준 템플릿 라이브러리(STL)

# 제네릭 클래스 만들기

2

## ■ 제네릭 클래스 선언

```
template <class T>
class MyStack {
    int tos;
    T data [100]; // T 타입의 배열
public:
    MyStack();
    void push(T element);
    T pop();
};
```

## ■ 제네릭 클래스 구현

```
template <class T>
void MyStack<T>::push(T element) {
    ...
}
template <class T> T MyStack<T>::pop() {
    ...
}
```

## ■ 클래스 구체화 및 객체 활용

```
MyStack<int> iStack; // int 타입을 다루는 스택 객체 생성
MyStack<double> dStack; // double 타입을 다루는 스택 객체 생성

iStack.push(3);
int n = iStack.pop();

dStack.push(3.5);
double d = dStack.pop();
```

# 예제 10-6 제네릭 스택 클래스 만들기

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

template <class T>
class MyStack {
    int tos;// top of stack
    T data [100]; // T 타입의 배열. 스택의 크기는 100
public:
    MyStack();
    void push(T element); // element를 data [] 배열에 삽입
    T pop(); // 스택의 탑에 있는 데이터를 data[] 배열에서 리턴
};

template <class T>
MyStack<T>::MyStack() { // 생성자
    tos = -1; // 스택은 비어 있음
}

template <class T>
void MyStack<T>::push(T element) {
    if(tos == 99) {
        cout << "stack full";
        return;
    }
    tos++;
    data[tos] = element;
}

template <class T>
T MyStack<T>::pop() {
    T retData;
    if(tos == -1) {
        cout << "stack empty";
        return 0; // 오류 표시
    }
    retData = data[tos--];
    return retData;
}
```

```
int main() {
    MyStack<int> iStack; // int 만 저장하는 스택
    iStack.push(3);
    cout << iStack.pop() << endl;

    MyStack<double> dStack; // double 만 저장하는 스택
    dStack.push(3.5);
    cout << dStack.pop() << endl;

    MyStack<char> *p = new MyStack<char>(); // char만 저장하는 스택
    p->push('a');
    cout << p->pop() << endl;
    delete p;
}
```

```
3
3.5
a
```

# 예제 10-7 두 개의 제네릭 타입을 가진 클래스 만들기

4

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

```
template <class T1, class T2> // 두 개의 제네릭 타입 선언
class GClass {
```

```
    T1 data1;
    T2 data2;
public:
    GClass();
    void set(T1 a, T2 b);
    void get(T1 &a, T2 &b);
};
```

data1을 a에, data2를  
b에 리턴하는 함수

```
template <class T1, class T2>
GClass<T1, T2>::GClass() {
    data1 = 0; data2 = 0;
}
```

```
template <class T1, class T2>
void GClass<T1, T2>::set(T1 a, T2 b) {
    data1 = a; data2 = b;
}
```

```
template <class T1, class T2>
void GClass<T1, T2>::get(T1 &a, T2 &b) {
    a = data1; b = data2;
}
```

```
int main() {
    int a;
    double b;
    GClass<int, double> x;
    x.set(2, 0.5);
    x.get(a, b);
    cout << "a=" << a << '\t' << "b=" << b << endl;

    char c;
    float d;
    GClass<char, float> y;
    y.set('m', 12.5);
    y.get(c, d);
    cout << "c=" << c << '\t' << "d=" << d << endl;
}
```

a=2	b=0.5
c=m	d=12.5

# C++ 표준 템플릿 라이브러리, STL

5

- STL(Standard Template Library)
  - ▣ 표준 템플릿 라이브러리
    - C++ 표준 라이브러리 중 하나
  - ▣ 많은 제네릭 클래스와 제네릭 함수 포함
    - 개발자는 이들을 이용하여 쉽게 응용 프로그램 작성
- STL의 구성
  - ▣ 컨테이너 – 템플릿 클래스
    - 데이터를 담아두는 자료 구조를 표현한 클래스
    - 리스트, 큐, 스택, 맵, 셋, 벡터
  - ▣ iterator – 컨테이너 원소에 대한 포인터
    - 컨테이너의 원소들을 순회하면서 접근하기 위해 만들어진 컨테이너 원소에 대한 포인터
  - ▣ 알고리즘 – 템플릿 함수
    - 컨테이너 원소에 대한 복사, 검색, 삭제, 정렬 등의 기능을 구현한 템플릿 함수
    - 컨테이너의 멤버 함수 아님

〈표 10-1〉 STL 컨테이너의 종류

컨테이너 클래스	설명	헤더 파일
vector	가변 크기의 배열을 일반화한 클래스	<vector>
deque	앞뒤 모두 입력 가능한 큐 클래스	<deque>
list	빠른 삽입/삭제 가능한 리스트 클래스	<list>
set	정렬된 순서로 값을 저장하는 집합 클래스, 값은 유일	<set>
map	(key, value) 쌍을 저장하는 맵 클래스	<map>
stack	스택을 일반화한 클래스	<stack>
queue	큐를 일반화한 클래스	<queue>

〈표 10-2〉 STL iterator의 종류

iterator의 종류	iterator에 ++ 연산 후 방향	read/write
iterator	다음 원소로 전진	read/write
const_iterator	다음 원소로 전진	read
reverse_iterator	지난 원소로 후진	read/write
const_reverse_iterator	지난 원소로 후진	read

〈표 10-3〉 STL 알고리즘 함수들

copy	merge	random	rotate
equal	min	remove	search
find	move	replace	sort
max	partition	reverse	swap

# STL과 관련된 헤더 파일과 이름 공간

7

## □ 헤더파일

### ▣ 컨테이너 클래스를 사용하기 위한 헤더 파일

- 해당 클래스가 선언된 헤더 파일 include

예) vector 클래스를 사용하려면 `#include <vector>`

list 클래스를 사용하려면 `#include <list>`

### ▣ 알고리즘 함수를 사용하기 위한 헤더 파일

- 알고리즘 함수에 상관 없이 `#include <algorithm>`

## □ 이름 공간

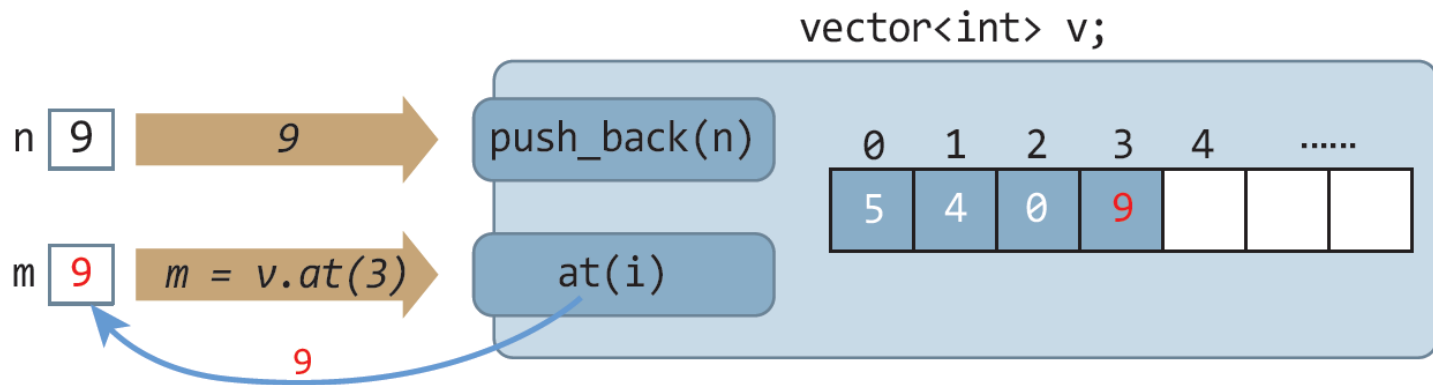
- ▣ STL이 선언된 이름 공간은 `std`

# vector 컨테이너

8

## □ 특징

- 가변 길이 배열을 구현한 제네릭 클래스
  - 개발자가 벡터의 길이에 대한 고민할 필요 없음
- 원소의 저장, 삭제, 검색 등 다양한 멤버 함수 지원
- 벡터에 저장된 원소는 인덱스로 접근 가능
  - 인덱스는 0부터 시작





# vector 클래스의 주요 멤버와 연산자

9

멤버와 연산자 함수	설명
<code>push_back(element)</code>	벡터의 마지막에 <code>element</code> 추가
<code>at(int index)</code>	<code>index</code> 위치의 원소에 대한 참조 리턴
<code>begin()</code>	벡터의 첫 번째 원소에 대한 참조 리턴
<code>end()</code>	벡터의 끝(마지막 원소 다음)을 가리키는 참조 리턴
<code>empty()</code>	벡터가 비어 있으면 <code>true</code> 리턴
<code>erase(iterator it)</code>	벡터에서 <code>it</code> 가 가리키는 원소 삭제. 삭제 후 자동으로 벡터 조절
<code>insert(iterator it, element)</code>	벡터 내 <code>it</code> 위치에 <code>element</code> 삽입
<code>size()</code>	벡터에 들어 있는 원소의 개수 리턴
<code>operator[]()</code>	지정된 원소에 대한 참조 리턴
<code>operator=()</code>	이 벡터를 다른 벡터에 치환(복사)

# vector 다루기 사례

vector 생성

```
vector<int> v;
```

정수 벡터  
생성

vector<int> v



정수 원소 삽입

```
v.push_back(1);  
v.push_back(2);  
v.push_back(3);
```

정수 삽입

v



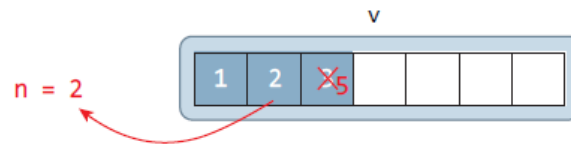
원소 개수 s  
벡터의 용량 c

```
int s = v.size(); // s는 3  
int c = v.capacity(); // c는 7
```

s = 3  
c = 7

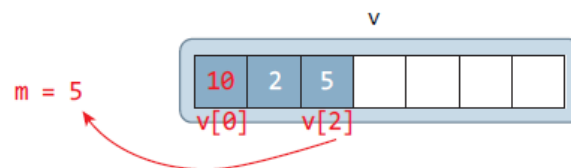
원소 값 접근

```
v.at(2) = 5;  
int n = v.at(1);
```



원소 값 접근

```
v[0] = 10;  
int m = v[2]; // m은 5
```



# 예제 10-9 vector 컨테이너 활용하기

11

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

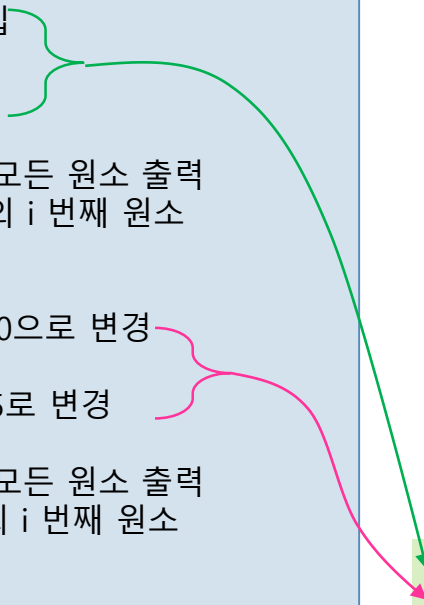
int main() {
    vector<int> v; // 정수만 삽입 가능한 벡터 생성

    v.push_back(1); // 벡터에 정수 1 삽입
    v.push_back(2); // 벡터에 정수 2 삽입
    v.push_back(3); // 벡터에 정수 3 삽입

    for(int i=0; i<v.size(); i++) // 벡터의 모든 원소 출력
        cout << v[i] << " "; // v[i]는 벡터의 i 번째 원소
    cout << endl;

    v[0] = 10; // 벡터의 첫 번째 원소를 10으로 변경
    int n = v[2]; // n에 3이 저장
    v.at(2) = 5; // 벡터의 3 번째 원소를 5로 변경

    for(int i=0; i<v.size(); i++) // 벡터의 모든 원소 출력
        cout << v[i] << " "; // v[i]는 벡터의 i 번째 원소
    cout << endl;
}
```



1 2 3  
10 2 5

# 예제 10-10 문자열을 저장하는 벡터 만들기 연습

12

string 타입의 vector를 이용하여 문자열을 저장하는 벡터를 만들고, 5개의 이름을 입력 받아 사전에 서 가장 뒤에 나오는 이름을 출력하라

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

```
int main() {
```

```
}
```

이름을 5개 입력하라

1>>황기태

2>>이재문

3>>김남윤

4>>한원선

5>>애슐리

사전에서 가장 뒤에 나오는 이름은 황기태

# iterator 사용

13

## □ iterator란?

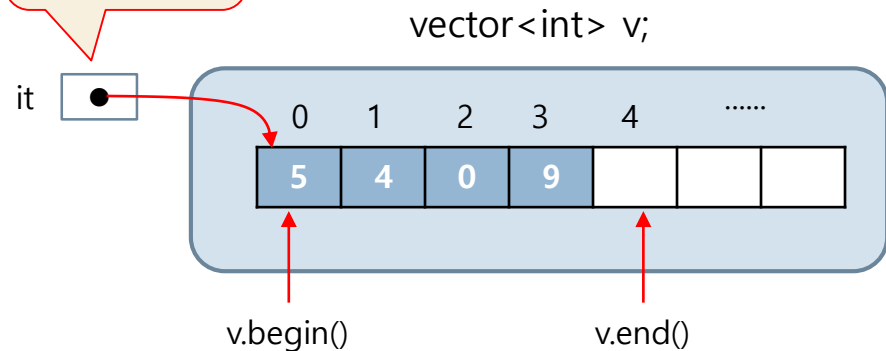
- 반복자라고도 부름
- 컨테이너의 원소를 가리키는 포인터

## □ iterator 변수 선언

- 구체적인 컨테이너를 지정하여 반복자 변수 생성

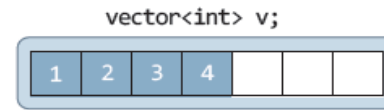
```
vector<int>::iterator it;  
it = v.begin();
```

it는 원소가 int  
타입인 벡터의  
원소에 대한 포  
인터



벡터 생성

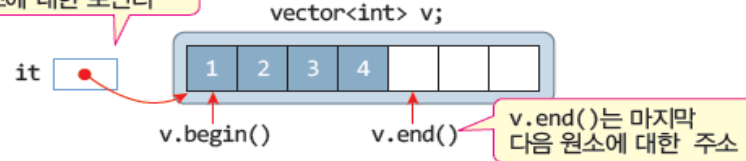
```
vector<int> v;  
for(int i=1; i<=4; i++)  
    v.push_back(i);
```



iterator 변수 선언  
및 초기화

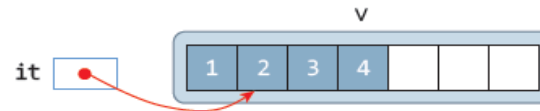
```
vector<int>::iterator it;  
it = v.begin();
```

it는 int 타입 벡터의  
원소에 대한 포인터



iterator 증가

```
it++;
```



원소 읽기

```
int n = *it;
```

n = 2

원소 쓰기

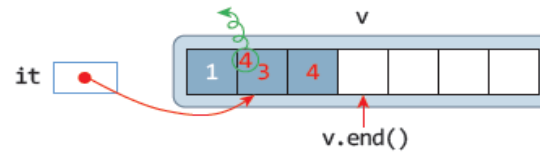
```
n = n*2;  
*it = n;
```



원소 삭제

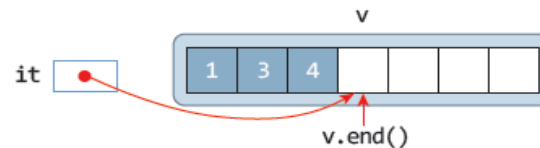
```
it = v.erase(it);
```

v.erase(it)는 it가 가리키는 원소를  
삭제한 후 다음 원소에 대한 포인터 리턴



끝으로 옮기기

```
it = v.end();
```



# 예제 10-11 iterator를 사용하여 vector의 모든 원소에 2 곱하기

15

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

int main() {
    vector<int> v; // 정수 벡터 생성
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);

    vector<int>::iterator it; // 벡터 v의 원소에 대한 포인터 it 선언

    for(it=v.begin(); it != v.end(); it++) { // iterator를 이용하여 모든 원소 탐색
        int n = *it; // it가 가리키는 원소 값 리턴
        n = n*2; // 곱하기 2
        *it = n; // it가 가리키는 원소에 값 쓰기
    }

    for(it=v.begin(); it != v.end(); it++) // 벡터 v의 모든 원소 출력
        cout << *it << ' ';
    cout << endl;
}
```

# map 컨테이너

16

## □ 특징

- ('키', '값')의 쌍을 원소로 저장하는 제네릭 컨테이너
  - 동일한 '키'를 가진 원소가 중복 저장되면 오류 발생
- '키'로 '값' 검색
- 많은 응용에서 필요함
- #include <map> 필요

## □ 맵 컨테이너 생성 예

- 영한 사전을 저장하기 위한 맵 컨테이너 생성 및 활용
  - 영어 단어와 한글 단어를 쌍으로 저장하고, 영어 단어로 검색

```
// 맵 생성
Map<string, string> dic;           // 키는 영어 단어, 값은 한글 단어

// 원소 저장
dic.insert(make_pair("love", "사랑")); // ("love", "사랑") 저장
dic["love"] = "사랑";                 // ("love", "사랑") 저장

// 원소 검색
string kor = dic["love"];             // kor은 "사랑"
string kor = dic.at("love");          // kor은 "사랑"
```



# map 클래스의 주요 멤버와 연산자

17

멤버와 연산자 함수	설명
<code>insert(pair&lt;&gt; &amp;element)</code>	맵에 '키'와 '값'으로 구성된 pair 객체 element 삽입
<code>at(key_type&amp; key)</code>	맵에서 '키' 값에 해당하는 '값' 리턴
<code>begin()</code>	맵의 첫 번째 원소에 대한 참조 리턴
<code>end()</code>	맵의 끝(마지막 원소 다음)을 가리키는 참조 리턴
<code>empty()</code>	맵이 비어 있으면 true 리턴
<code>find(key_type&amp; key)</code>	맵에서 '키' 값에 해당하는 원소를 가리키는 iterator 리턴
<code>erase(iterator it)</code>	맵에서 it가 가리키는 원소 삭제
<code>size()</code>	맵에 들어 있는 원소의 개수 리턴
<code>operator[key_type&amp; key]()</code>	맵에서 '키' 값에 해당하는 원소를 찾아 '값' 리턴
<code>operator=()</code>	맵 치환(복사)

# 예제 10-12 map으로 영한 사전 만들기

18

map 컨테이너를 이용하여  
(영어, 한글) 단어를 쌍으로  
저장하고, 영어로 한글을 검  
색하는 사전을 작성하라.

```
저장된 단어 개수 3
찾고 싶은 단어>> apple
사과
찾고 싶은 단어>> lov
없음
찾고 싶은 단어>> love
사랑
찾고 싶은 단어>> exit
종료합니다...
```

```
#include <iostream>
#include <string>
#include <map>
using namespace std;

int main() {
    map<string, string> dic; // 맵 컨테이너 생성. 키는 영어 단어, 값은 한글 단어

    // 단어 3개를 map에 저장
    dic.insert(make_pair("love", "사랑")); // ("love", "사랑") 저장
    dic.insert(make_pair("apple", "사과")); // ("apple", "사과") 저장
    dic["cherry"] = "체리"; // ("cherry", "체리") 저장

    cout << "저장된 단어 개수 " << dic.size() << endl;

    string eng;
    while (true) {
        cout << "찾고 싶은 단어>> ";
        getline(cin, eng); // 사용자로부터 키 입력
        if (eng == "exit")
            break; // "exit"이 입력되면 종료

        if(dic.find(eng) == dic.end()) // eng '키'를 끝까지 찾았는데 없음
            cout << "없음" << endl;
        else
            cout << dic[eng] << endl; // dic에서 eng의 값을 찾아 출력
    }
    cout << "종료합니다..." << endl;
}
```

# STL 알고리즘 사용하기

19

## □ 알고리즘 함수

- ▣ 템플릿 함수

- ▣ 전역 함수

  - STL 컨테이너 클래스의 멤버 함수가 아님

- ▣ iterator와 함께 작동

## □ sort() 함수 사례

- ▣ 두 개의 매개 변수

  - 첫 번째 매개 변수 : 소팅을 시작한 원소의 주소

  - 두 번째 매개 변수 : 소팅 범위의 마지막 원소 다음 주소

```
vector<int> v;
```

```
...
```

```
sort(v.begin(), v.begin()+3); // v.begin()에서 v.begin()+2까지, 처음 3개 원소 정렬
```

```
sort(v.begin()+2, v.begin()+5); // 벡터의 3번째 원소에서 v.begin()+4까지, 3개 원소 정렬
```

```
sort(v.begin(), v.end()); // 벡터 전체 정렬
```

# 예제 10-13 sort() 함수를 이용한 vector 소팅

20

정수 벡터에 5개의 정수를 입력 받아 저장하고, sort()를 이용하여 정렬하는 프로그램을 작성하라.

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

int main() {
    vector<int> v; // 정수 벡터 생성

    cout << "5개의 정수를 입력하세요>> ";
    for(int i=0; i<5; i++) {
        int n;
        cin >> n;
        v.push_back(n); // 키보드에서 읽은 정수를 벡터에 삽입
    }

    // v.begin()에서 v.end() 사이의 값을 오름차순으로 정렬
    // sort() 함수의 실행 결과 벡터 v의 원소 순서가 변경됨
    sort(v.begin(), v.end());

    vector<int>::iterator it; // 벡터 내의 원소를 탐색하는 iterator 변수 선언

    for(it=v.begin(); it != v.end(); it++) // 벡터 v의 모든 원소 출력
        cout << *it << ' ';
    cout << endl;
}
```

주목

5개의 정수를 입력하세요>> 30 -7 250 6 120  
-7 6 30 120 250

# 람다

21

## □ 람다 대수와 람다식

- ▣ 람다 대수에서 람다식은 수학 함수를 단순하게 표현하는 기법

$x, y$ 를 더하는 수학 함수  $f$

$$f(x, y) = x + y$$



함수  $f$ 의 람다식

$$(x, y) \rightarrow x + y$$

람다식  $f$  계산

$$\begin{aligned} & ((x, y) \rightarrow x + y)(2, 3) \\ &= 2 + 3 \\ &= 5 \end{aligned}$$

## □ C++ 람다

- ▣ 익명의 함수 만드는 기능으로 C++11에서 도입
  - 람다식, 람다 함수로도 불림
  - C#, Java, 파이썬, 자바스크립트 등 많은 언어들이 도입하고 있음

# C++에서 람다식 선언

22

## □ C++의 람다식의 구성

### ▣ 4 부분으로 구성

- 캡처 리스트 : 람다식에서 사용하고자 하는 함수 바깥의 변수 목록
- 매개변수 리스트 : 보통 함수의 매개변수 리스트와 동일
- 리턴 타입
- 함수 바디 : 람다식의 함수 코드

[ ] ( ) -> 리턴타입 { /\* 함수 코드 작성 \*/ };  
The diagram uses brackets and arrows to identify parts of the lambda expression: [ ] is labeled '캡처 리스트' (capture list), ( ) is labeled '매개변수 리스트' (parameter list), -> is labeled '생략 가능' (optional), and { } is labeled '함수 바디' (function body). The text '리턴타입' (return type) is placed between the parameter list and the function body.

(a) 람다식의 기본 구조

```
[ ](int x, int y) { cout << x + y; }; // 매개변수 x, y의 합을 출력하는 람다 작성  
[ ](int x, int y) -> int { return x + y; }; // 매개변수 x, y의 합을 리턴하는 람다 작성  
[ ](int x, int y) { cout << x + y; } (2, 3); // x에 2, y에 3을 대입하여 코드 실행. 5 출력
```

(b) 람다식 작성 및 호출 사례

# 간단한 람다식 만들기

23

## 예제 10-15 매개변수 x, y의 합을 출력하는 람다식 만들기

매개변수 x, y의 합을 출력하는 람다식은 다음과 같이 작성

```
[](int x, int y) { cout << x + y; }; // x, y의 합을 출력하는 람다식
```

x에 2, y에 3을 전달하여 람다식이 바로 실행된다.

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

int main() {
    // 람다 함수 선언과 동시에 호출(x=2, y=3 전달)
    [](int x, int y) { cout << "합은 " << x + y; } (2, 3); // 5 출력
}
```

합은 5

# 캡처 리스트와 리턴 타입을 가지는 람다식

24

## 예제 10-17 반지름이 r이 원의 면적으로 리턴하는 람다식 만들기

지역 변수 pi의 값을 받고, 매개변수 r을 이용하여 반지름 값을 전달받아, 원의 면적을 계산하여 리턴하는 람다식을 작성하고, 람다식을 호출하는 코드를 프로그램을 작성하라.

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

int main() {
    double pi = 3.14; // 지역 변수

    auto calc = [pi](int r) -> double { return pi*r*r; };

    cout << "면적은 " << calc(3); // 람다식 호출. 28.26출력
}
```

람다식

면적은 28.26

\* 캡처 리스트와 리턴타입을 가지는 람다식 연습



# 캡처 리스트에 참조를 활용하는 람다식

25

## 예제 10-18 캡처 리스트에 참조 활용. 합을 외부에 저장하는 람다식 만들기

지역 변수 `sum`에 대한 참조를 캡처 리스트를 통해 받고, 합한 결과를 지역변수 `sum`에 저장한다.

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

int main() {
    int sum = 0; // 지역 변수

    [&sum](int x, int y) { sum = x + y; } (2, 3); // 합 5를 지역변수 sum에 저장

    cout << "합은 " << sum;
}
```

합은 5

\* 캡처 리스트를 통해 지역 변수의 참조를 받아 지역 변수를 접근하는 연습