



# 고급프로그래밍

## 람다식

Professor Jeong, Mun-Ho

Robot Vision & Intelligence Laboratory  
Kwangwoon University  
(02-940-5625, mhjeong@kw.ac.kr)

# Schedule

주차	주제		과제	퀴즈
1	과목소개	교과목 소개 (1), C++ 시작 (2)		
2	C++	C++ 프로그래밍의 기본 (3), 클래스와 객체 (4)	1	1
3		객체생성과 사용 (5)	2	2
4		함수와 참조 (6, 3/26), 복사 생성자와 함수중복(7)	3	3
5		static, friend, 연산자중복 (8, 4/2), 연산자중복 상속(9)	4	4
6		상속 (10, 4/9), 가상함수와 추상클래스 (11)		5
7		템플릿과 STL (12, 4/16), 입출력(13)	5	
8		중간고사		
9		파일 입출력(14), 예외처리 및 C 사용(15)		6
10		람다식(16, 5/7) , 멀티스레딩	6	
11		멀티스레딩, 고급문법		
12		고급문법		
13	병렬프로그래밍	병렬프로그래밍		
14		병렬프로그래밍		
15		기말고사		

# 오늘의 학습내용

- 람다식

람다식

# auto를 이용하여 쉬운 변수 선언

## ■ C++에서 auto

- 컴파일러에게 변수선언문에서 추론하여 타입을 자동 선언하도록 지시
- 복잡한 변수 선언의 간소화

## ■ auto의 기본 사용 사례

```
auto pi = 3.14;    // 3.14가 실수이므로 pi는 double 타입으로 선언됨  
auto n = 3;        // 3이 정수이므로 n을 int 타입으로  
auto *p = &n;      // 변수 p는 int* 타입으로 추론
```

```
int n = 10;  
int &ref = n;       // ref는 int에 대한 참조 변수  
auto ref2 = ref;    // ref2는 int& 변수로 자동 선언
```

# auto의 다른 활용 사례

## ■ 다른 활용 사례

- 함수의 리턴 타입으로부터 추론하여 변수 타입 선언

```
int square(int x) { return x*x; }  
...  
auto ret = square(3); // 변수 ret는 int 타입으로 추론
```

- STL 템플릿에 활용

- `vector<int>::iterator` 타입의 변수 `it`를 `auto`를 이용하여 간단히 선언

```
vector<int>::iterator it;  
  
for (it = v.begin(); it != v.end(); it++)  
    cout << *it << endl;
```



```
for (auto it = v.begin(); it != v.end(); it++)  
    cout << *it << endl;
```

# auto의 다른 활용 사례

## ■ 다른 활용 사례

- 함수의 리턴 타입으로부터 추론하여 변수 타입 선언

```
int square(int x) { return x*x; }  
...  
auto ret = square(3); // 변수 ret는 int 타입으로 추론
```

- STL 템플릿에 활용

- `vector<int>iterator` 타입의 변수 `it`를 `auto`를 이용하여 간단히 선언

```
:  
int main()  
{  
    vector<int> buff(10); // 원소를 10개 생성  
  
    for (auto x : buff)  
        cout << x << endl;  
}
```

# 예제

```
:
int square(int x) { return x*x; }

int main() {
    // 기본 타입 선언에 auto 활용
    auto c = 'a';      // c는 char 타입으로 결정
    auto pi = 3.14;    // pi는 double 타입으로 결정
    auto ten = 10;     // ten은 int 타입으로 결정
    auto *p = &ten;    // 변수 p는 int* 타입으로 결정
    cout << c << " " << pi << " " << ten << " " << *p << endl;

    // 함수의 리턴 타입으로 추론
    auto ret = square(3); // square() 함수의 리턴 타입이 int 이므로 ret는 int로 결정
    cout << *p << " " << ret << endl;

    vector<int> v = { 1,2,3,4, 5 }; //벡터 v에 5개의 원소, 1,2,3,4,5 삽입
    vector<int>::iterator it;
    for (it = v.begin(); it != v.end(); it++)
        cout << *it << " "; // 1 2 3 4 5 출력
    cout << endl;

    // 템플릿에 auto를 사용하여 간소화
    for (auto x : v)
        cout << x << " "; // 1 2 3 4 5 출력
}
```

```
a 3.14 10 10
10 9
1 2 3 4 5
1 2 3 4 5
```



# 람다

## ■ 람다 대수와 람다식

- 람다 대수에서 람다식은 수학 함수를 단순하게 표현하는 기법

x,y를 더하는 수학 함수 f

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



함수 f의 람다식

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

람다식 f 계산

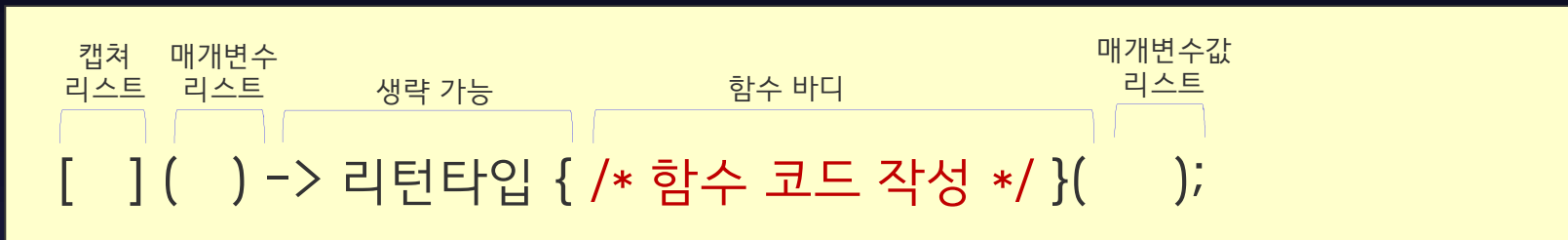
$((x, y) \rightarrow x + y)(2, 3)$   
 $= 2 + 3$   
 $= 5$

## ■ C++ 람다

- 익명의 함수 만드는 기능으로 C++11에서 도입
  - 람다식, 람다 함수로도 불림
  - C#, Java, 파이썬, 자바스크립트 등 많은 언어들이 도입하고 있음
- 인라인 함수처럼 처리 속도 빠름
- 코드 간소화

# C++에서 람다식 선언

## ■ C++의 람다식의 구성



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

## ■ 람다식 호출 예

```
[ ](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
```

# 예제

- 매개변수 x, y의 합을 출력
  - x에 2, y에 3을 전달하여 람다식이 바로 실행

```
#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

# Capture 리스트

1. `[&]() { /* */ }` 외부의 모든 변수들을 레퍼런스로 가져온다
2. `[=]() { /* */ }` 외부의 모든 변수들을 값으로 가져온다
3. `[=, &x, &y] { /* */ }`, 혹은 `[&, x, y] { /* */ }` 외부의 모든 변수들을 값/레퍼런스로 가져오되, `x` 와 `y` 만 레퍼런스/값으로 가져온다
4. `[x, &y, &z] { /* */ }` 지정한 변수들을 지정한 바에 따라 가져온다.

# STL for\_each()

- STL에 들어 있는 for\_each() 함수는 컨테이너의 각 원소를 검색하는 함수이며, 3번째 매개변수로 주어진 함수를 호출한다.

```
#include <iostream>
#include <vector>
#include <algorithm> // for_each() 알고리즘 함수를 사용하기 위함
using namespace std;

void print(int n) {
    cout << n << " ";
}

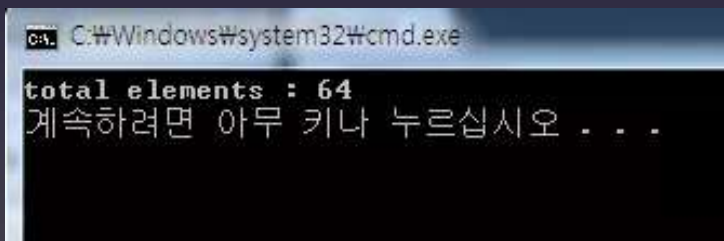
int main() {
    vector<int> v = { 1, 2, 3, 4, 5 };

    // for_each()는 벡터 v의 첫번째 원소부터 끝까지 검색하면서,
    // 각 원소에 대해 print(int n) 호출. 매개 변수 n에 각 원소 값 전달
    for_each(v.begin(), v.end(), print);
}
```

1 2 3 4 5

# 예제

```
:  
int main( )  
{  
    int total_elements = 1;  
    vector<int> cardinal;  
  
    cardinal.push_back(1);  
    cardinal.push_back(2);  
    cardinal.push_back(4);  
    cardinal.push_back(8);  
  
    for_each(cardinal.begin(), cardinal.end(), [&](int i) { total_elements *= i; });  
  
    cout << "total elements : " << total_elements << endl;  
}
```



C:\Windows\system32\cmd.exe

```
total elements : 64  
계속하려면 아무 키나 누르십시오 . . .
```

# auto로 람다식 저장 및 호출

- auto를 이용하여 변수 love에 람다식을 저장하고, love를 이용하여 람다식 호출

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

```
int main() {
    auto love = [ ](string a, string b) {
        cout << a << "보다 " << b << "가 좋아" << endl;
    };
}
```

람다식

```
love("돈", "너");    // 람다식 호출
love("냉면", "만두"); // 람다식 호출
```

```
돈보다 너가 좋아
냉면보다 만두가 좋아
```

- \* auto를 이용하여 람다식을 변수에 저장하는 사례
- \* 람다식의 형식은 컴파일러만 알기 때문에, 개발자가 람다식을 저장하는 변수의 타입을 선언할 수 없음!

# 예제

- 지역 변수 `sum`에 대한 참조를 캡처 리스트를 통해 받고, 합한 결과를 지역변수 `sum`에 저장

```
#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



# 실습

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

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

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

    auto calc = 

    cout << "Area : " << 
}
```

Area : 28.26

# 템플릿과 람다식

```
#include <iostream>
#include <functional>

using namespace std;

template<class T>
int MinValue(const int npA[], int nNum, T fun)
{
    int nMin = fun(npA[0]);
    for(int i=0; i<nNum; i++)
        if(fun(npA[i] < nMin))
            nMin = fun(npA[i]);
    return nMin;
}

int main()
{
    int npArray[5] = { 7, -5, 9, -2, 3};

    cout << MinValue(npArray, 5, [](int n){ return n*n; }) << endl;
    cout << MinValue(npArray, 5, [](int n){ return n*(n-5); }) << endl;
}
```

# Capture는 람다식이 정의될 때

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

int main()
{
    int v = 42;
    auto func = [=] { cout << v << endl; };
    v = 8;
    func();
}
```

42

# 값의 Capture는 자동 const

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

int main()
{
    int i = 10;
    auto two_i = [=] {
        i *= 2;    //compile error
        return i;
    };
    cout << "2i : " << two_i( ) << " i:" << i << endl;
}
```

# 값의 Capture는 자동 const

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

int main()
{
    int i = 10;
    auto tow_i = [=]( ) mutable ->int {
        i *= 2;    //함수 내부변수 i를 2배
        return i;
    };
    cout << "2i : " << tow_i( ) << " i:" << i << endl;
}
```

# Capture의 범위

```
:  
  
int main()  
{  
    int i = 8;  
    auto func = [i]() {  
        int j = 2;  
        auto m = [=]() { cout << i / j; } ;  
        m();  
    };  
    func();  
}
```

```
:  
  
int main()  
{  
    int i = 8;  
    auto func = [ ]() {  
        int j = 2;  
        auto m = [=]() { cout << i / j; } ;  
        m();  
    };  
    func();  
}
```

```
:  
  
int main()  
{  
    int i = 8;  
    auto func = [=]() {  
        int j = 2;  
        auto m = [&]() { i /= j; } ;  
        m();  
    };  
    func();  
}
```

# 예제 - STL for\_each()

- STL에 들어 있는 for\_each() 함수는 컨테이너의 각 원소를 검색하는 함수이며, 3번째 매개변수로 주어진 함수를 호출한다.

```
#include <iostream>
#include <vector>
#include <algorithm> // for_each() 알고리즘 함수를 사용하기 위함
using namespace std;

void print(int n) {
    cout << n << " ";
}

int main() {
    vector<int> v = { 1, 2, 3, 4, 5 };

    // for_each()는 벡터 v의 첫번째 원소부터 끝까지 검색하면서,
    // 각 원소에 대해 print(int n) 호출. 매개 변수 n에 각 원소 값 전달
    for_each(v.begin(), v.end(), print);
}
```

1 2 3 4 5


# 실습 - STL 템플릿에 람다식 활용

- STL에 들어 있는 for-each() 함수는 컨테이너의 각 원소를 검색하는 함수이며, 3번째 매개변수로 주어진 함수를 호출한다.

```
#include <iostream>
#include <vector>
#include <algorithm> // for_each() 알고리즘 함수를 사용하기 위함
using namespace std;

int main() {
    vector<int> v = { 1, 2, 3, 4, 5 };

    for_each(v.begin(), v.end(),  )
}
```



C:\Qt\Tools\QtCreator\bin\qtcreator\_process\_stub.exe

```
1 2 3 4 5
Press <RETURN> to close this window...
```



