객체지향 프로그래밍

3 차시

- 리뷰 (변수, 함수, 조건문)
- 배열
- 반복문

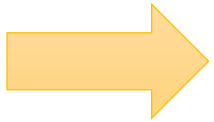
Review - 헤더

- C/C++에서 다양한 함수들을 사용하기 위해, 코드에 포함시키는 모듈
- 입출력을 하기 위해 우리는 지난 시간 iostream 헤더를 include 하였음

Review – main 함수

- C/C++ 프로그램은 main 함수의 첫 줄을 읽으며 시작되고, main 함수가 반환을 하면 프로그램은 종료됨

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

int main() {
    int a;
    a = 3;

    return 0;
    cout << a << endl;
}
```



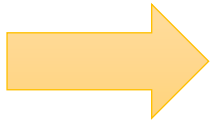
콘솔

프로그램이 종료되었습니다. ㅋㅋ

Review – 세미콜론

- C/C++ 코드는 한 줄의 끝을 세미콜론(;)으로 판별함
  - 세미콜론을 항상 붙입시다

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

int main() {
    int a
    a = 3

    return 0;
}
```



오류 목록

';'가 필요합니다.

Review – 변수

- 사실상 선언의 정의가 모호했던 파이썬과 달리, C/C++는 변수의 선언과 초기화가 명확함
- 변수는 반드시 선언을 하여야 사용할 수 있음
- 변수를 선언할 때는 자료형과 변수이름을 명시하여야 함

```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5   int a; // 선언
6   a = 1; // 초기화
7   int b = 2; // 선언, 초기화 동시에
8
9   int c, d; // 선언 여러개
10  c = 3, d = 4; // 초기화 동시에 여러개
11  int e = 5, f = 6; // 선언과 초기화 동시에 여러개
12 }
```

변수 선언의 기본형

자료형 변수이름;

Review – 입출력

- 출력은 cin(console in), 입력은 cout(console out)
- <<(Left bit shift operator), >>(Right bit shift operator)를 이용함
  - 위의 연산자는 원래의 용도와는 쓰임새가 전혀 다르지만, C++ 개발자가 일관성을 포기하고 직관성을 취하면서 Shift 연산자를 채용하였음

변수 출력

cout << 변수;

변수에서 콘솔로

변수 입력

cin >> 변수;

콘솔에서 변수로

배열 시작

배열? in Python

- 실행결과는?

```
1 myList = list()
2
3 myList.append(1)
4 myList.append(2)
5
6 print(myList)
```

배열? in Python

- 실행결과는?

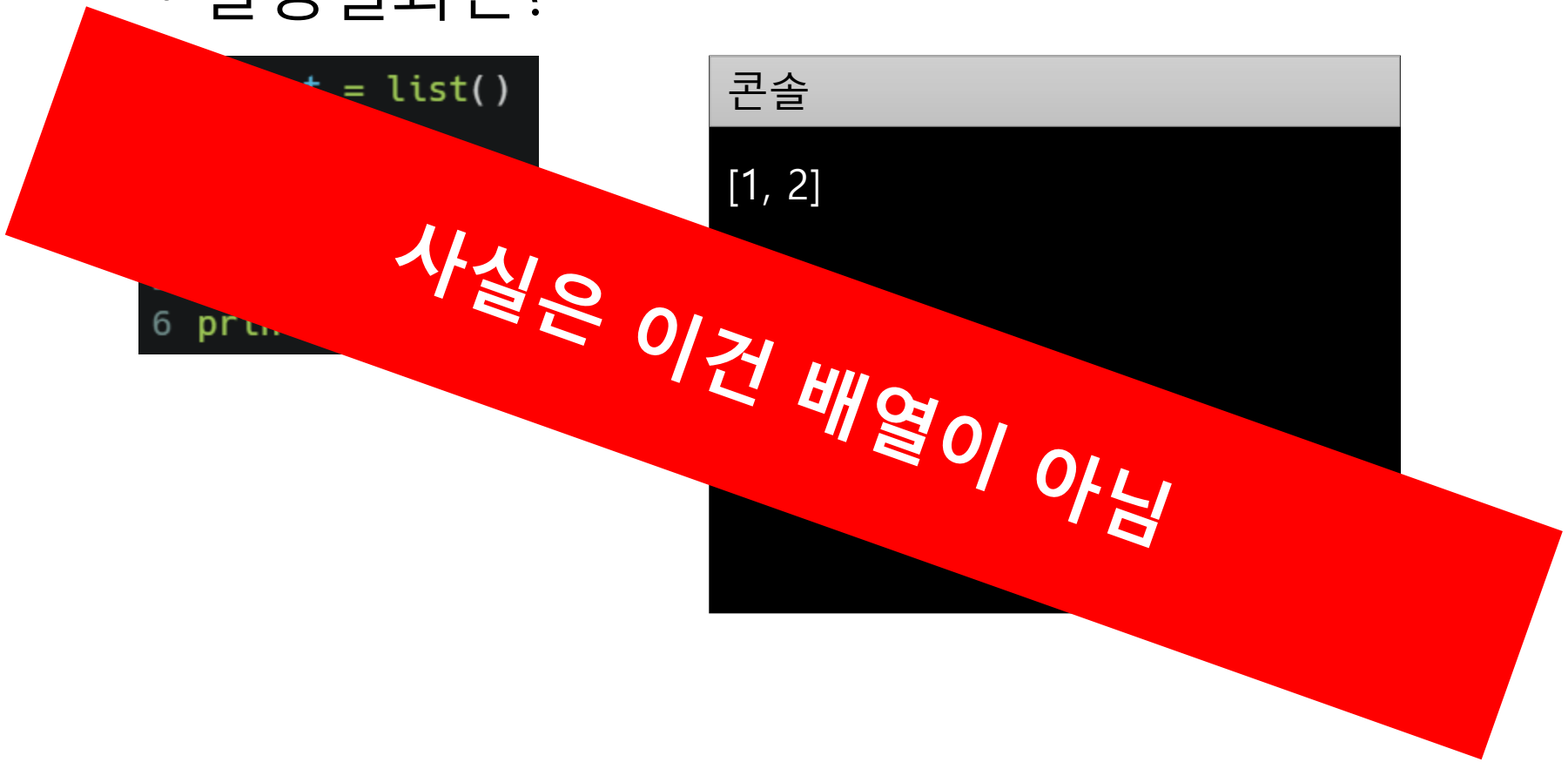
```
1 myList = list()
2
3 myList.append(1)
4 myList.append(2)
5
6 print(myList)
```

콘솔

```
[1, 2]
```

배열? in Python

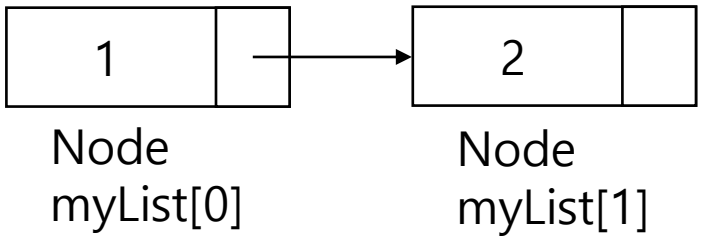
- 실행결과는?



리스트

- 리스트의 데이터 저장 원리

```
1 myList = list()
2
3 myList.append(1)
4 myList.append(2)
5
6 print(myList)
```



배열 in C++

- 실행 결과는?

```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5   int myArr[2];
6   myArr[0] = 1;
7   myArr[1] = 2;
8
9   cout << myArr[0] << endl;
10  cout << myArr[1] << endl;
11  cout << myArr << endl;
12 }
```

배열 in C++

- 실행 결과는?

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

int main(){
  int myArr[2];
  myArr[0] = 1;
  myArr[1] = 2;

  cout << myArr[0] << endl;
  cout << myArr[1] << endl;
  cout << myArr << endl;
}
```

콘솔

```
1
2
00000064D777F698
```

배열의 선언

- 배열은 함수와 변수들 처럼 선언과정이 필요함
- 배열의 자료형은 배열에 담을 자료형이랑 같음
  - 즉 다른 자료형을 요소에 담을 수 없음

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

int main(){
  int myArr[2];
  myArr[0] = 1;
  myArr[1] = 2;

  cout << myArr[0] << endl;
  cout << myArr[1] << endl;
  cout << myArr << endl;
}
```

배열의 초기화 – 요소 초기화

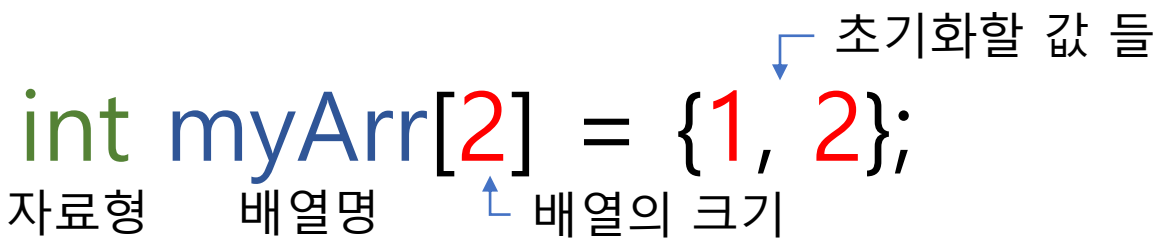
- 배열의 각각의 요소를 초기화 할 수 있음
- 배열의 자료형과 다른 값을 넣으면 원하지 않는 동작을 하게 됨

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

int main(){
  int myArr[2];
  myArr[0] = 1;
  myArr[1] = 2;

  cout << myArr[0] << endl;
  cout << myArr[1] << endl;
  cout << myArr << endl;
}
```

배열의 초기화 – 한번에 초기화

- 배열 전체를 한꺼번에 초기화할 수 있음
- 단, 선언과 초기화를 동시에 해야지만 가능함

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int myArr[2] = { 1, 2 };
6
7      cout << myArr[0] << endl;
8      cout << myArr[1] << endl;
9      cout << myArr << endl;
10 }
```

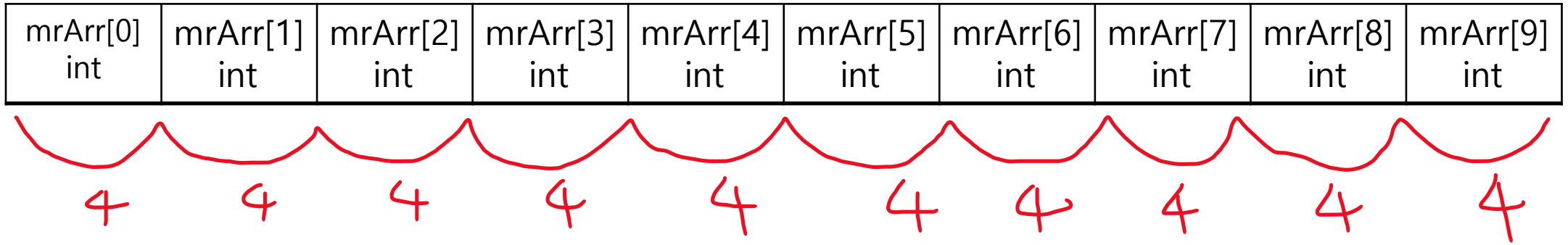


배열의 원리

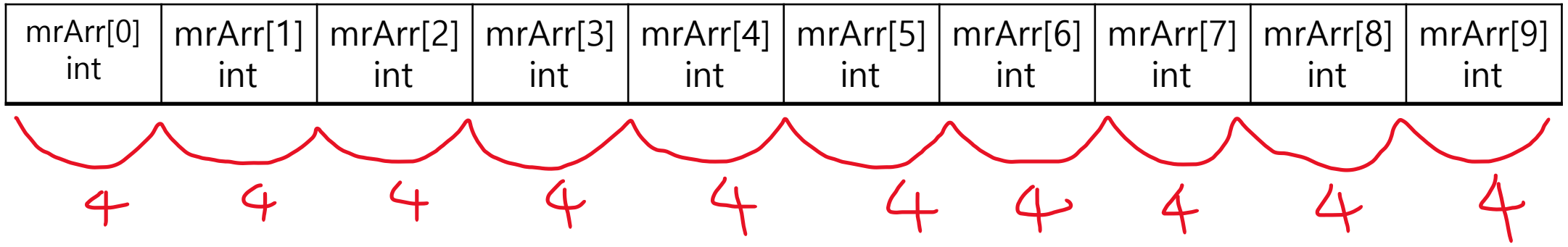
- 연속된 공간에 원하는 크기 만큼 변수를 할당하는 방식

```
int myArr[10];
```

- int형 변수 10개를 연속된 공간에 할당해주세요!



왜 연속된 공간이 필요하지?



- 배열의 이름은 배열의 시작 주소값을 담는 포인터 변수임
- 컴파일러는 배열의 인덱스를 탐색할 때
- 배열의 시작주소에 (자료형 byte * index)를 더하여 탐색함

mrArr[3] => mrArr + (4 * 3)

이차원 배열?

- 1차원 배열은 1차원 공간 상에 변수가 자리 했다면

mrArr[0] int	mrArr[1] int	mrArr[2] int	mrArr[3] int	mrArr[4] int	mrArr[5] int	mrArr[6] int	mrArr[7] int	mrArr[8] int	mrArr[9] int

- 2차원 배열은 2차원 공간 상에 변수가 자리한다고 생각하면 됨

mrArr[0] int	mrArr[1] int	mrArr[2] int	mrArr[3] int	mrArr[4] int
mrArr[5] int	mrArr[6] int	mrArr[7] int	mrArr[8] int	mrArr[9] int

이차원 배열?

- 1차원 배열은 1차원 공간 상에 변수가 자리 했다면

mrArr[0] int	mrArr[1] int	mrArr[2] int	mrArr[3] int	mrArr[4] int	mrArr[5] int	mrArr[6] int	mrArr[7] int	mrArr[8] int	mrArr[9] int

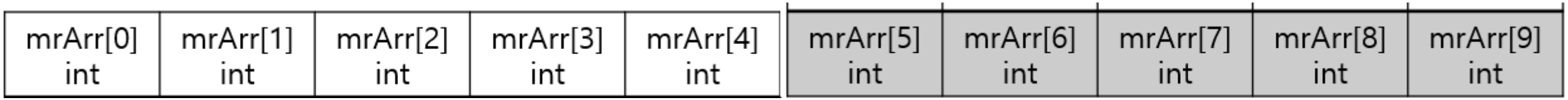
- 2차원 배열은 2차원 공간 상에 변수가 자리한다고 생각하면 됨

mrArr[0] int	mrArr[1] int	mrArr[2] int	mrArr[3] int	mrArr[4] int
mrArr[5] int	mrArr[6] int	mrArr[7] int	mrArr[8] int	mrArr[9] int

```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5   int myArr[2][5];
6 }
```

이차원 배열?

- 2차원 배열은 우리가 생각하기에 2차원 공간에 적재되어 있다고 느끼지만, 사실은 직선 상에 배치 되었음



- 그래서 배열을 한번에 초기화할 때 다음과 같이 함

```
int main() {
    int arr[2][5] = {1,2,3,4,5,6,7,8,9,10};
    cout << arr[1][3] << endl;

}
```

배열 vs 리스트

- 배열은 컴파일 당시에 크기가 정해져서 할당됨
- 리스트는 런타임 시간에 계속 요소를 추가하여 크기를 변경할 수 있음

- 그럼 리스트가 더 좋은거 아님?
  - 꼭 그렇지만은 않음. 프로그램에서 기능이 더 많다는 것은 더 많은 연산을 요구한다는 것이고, 이는 프로그램의 실행속도에 영향을 주게 됨
  - 자세한 내용은 자료구조에서...

문자열!

- Python을 배운 입장에서 C++를 할 때, 킹받는 부분 중 하나는 string 타입을 넣을 수 있는 자료형이 없다는 것임

- char 자료형은 문자 단 하나만을 저장할 수 있기 때문에 문자열을 표현하기에는 적합하지 않음

char 배열

- 편의상 ‘차 배열’이라고 부르겠음(나만 이렇게 부르는지 다른 사람도 이렇게 부르는지는 잘모르겠음)

- 우리는 char 자료형과 배열을 활용하여 문자열을 입력할 수 있음

char 배열

- 아래의 코드는 11개의 char 변수를 저장하는 배열을 선언하고 “HelloWorld” 문자열을 저장하였음

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     char myStr[11] = "HelloWorld";
6     cout << myStr << endl;
7 }
```

- 출력도 잘됨

char 배열

- 각각의 인덱스에는 한글자 한글자 글자들이 입력되고, 마지막 요소에는 NULL값이 들어감
- NULL값은 문자열의 끝을 의미하게 됨

H	e	l	l	o	W	o	r	l	d	NULL

- 위의 코드에서 11을 10으로 바꾸게 되면 컴파일 안됨 ㅋㅋ
  - why? 문자열의 끝을 의미하는 NULL이 들어갈 자리가 없기 때문

string 클래스

- 아 차 배열~ 개 불편하고 킹받네~
- 그런 당신을 위해 준비했습니다

#include <string>

string 클래스

- <string> 헤더를 추가하면 사용할 수 있음

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main() {
6     string a = "Hello, World!";
7     cout << a << endl;
8 }
```

- string은 클래스이지만, 자료형처럼 사용할 수 있음
  - 여기에서 객체지향 프로그래밍의 본질을 알 수 있음, 기능을 하는 무언가를 기본 문법으로 클래스의 형태로 구현하여 필요할 때 편하게 꺼내 쓸 수 있는..

string 클래스

- Python의 String처럼 다양한 메소드를 지원함

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

int main() {
    string a = "Hello, World!";
    cout << a << endl;
}
```

- https://blockdmask.tistory.com/338 << 여기에 좋은 내용 많음

반복문

- C++의 반복문은 크게 2가지로 나눌 수 있을 것 같음, 또 세분화 하면 2개로 다시 나눌 수 있음
  - for
    - 인덱스 기반 for
    - 요소 기반 for
  - while
    - standard while
    - do-while

반복문 – while

- 조건식이 true이면 반복하는 구문
- Python이랑 다를 게 없다!

```
int i = 0;
while (i < 10){
    i++;
    // 10번 반복 하는 반복문
}
```

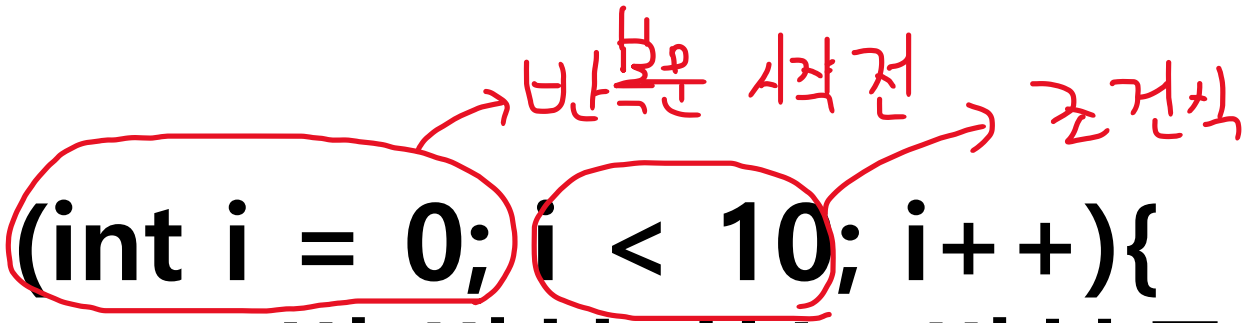
반복문 – do-while

- 무조건 “한 번은” 실행하는 반복문, 한번 실행한 후에는 while문과 동일하게 돌아감

```
int i = 0;
do{
        i++;
        // 10번 반복 하는 반복문
} while(i < 10);
```

반복문 – for (인덱스 기반)

- while 구문을 간결하게 사용할 때 쓰는 반복문, 편함!




```
for (int i = 0; i < 10; i++){
    // 10번 반복 하는 반복문
}
```

반복문 – for (요소 기반)

- 파이썬의 for와 같음(C++11에서 추가됨)

```
for (auto elem : arr)
    // arr의 요소를 하나하나 elem에
    담아서 사용하는 for문
}
```

```
1 arr = [1,2,3,4,5]
2 for elem in arr:
3   # 1,2,3,4,5 쓰는 for문
```

배열과 반복문

- 사실 배열과 반복문은 큰 연관이 있음
- 배열의 요소를 탐색할 때 반복문이 효과적임

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

int main(){
  int arr[5];
  for(int i = 0; i<5; i++){
    arr[i] = i+1;
  }
  for(auto elem:arr){
    cout << elem << endl;
  }
}
```

이차원 배열과 이중 반복문

- 우리는 글을 쓸 때 한줄한줄 적어나감,
- 그런데, 그 한 줄을 적을 때는 왼쪽에서 오른쪽으로 채워나감

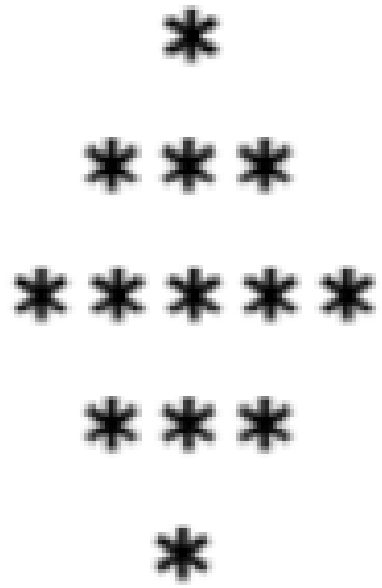
[0][0]

가	나	다	라	마
바	사	아	자	차
카	타	파	하	A
B	C	D	E	F
G	H	I	J	K

[4][4]

실습1

- 소위 찍기!
- 반복문을 사용하여 찍어보자



실습2

- string에는 split함수가 없다, split 함수를 직접 구현해보자