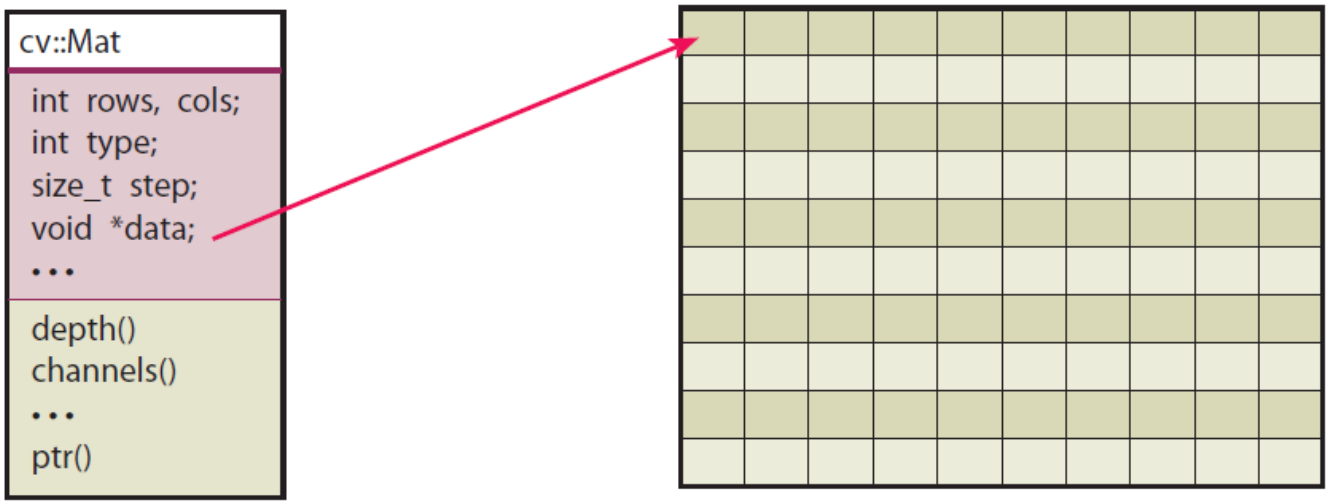
3장 OpenCV의 기초

Mat 클래스

Mat 클래스

- Mat 클래스는 OpenCV에서 영상을 담을 때 사용하는 데이터 구조로서 OpenCV 라이브러리의 핵심 요소



생성자

생성자의 인수 구조

```
Mat::Mat();
Mat::Mat(int rows, int cols, int type)
Mat::Mat(int rows, int cols, int type, const Scalar& s)
Mat::Mat(int rows, int cols, int type, void* data, size_t step = AUTO_STEP)
Mat::Mat(Size size, int type)
Mat::Mat(Size size, int type, const Scalar& s)
Mat::Mat(Size size, int type, void* data, size_t step = AUTO_STEP)

Mat::Mat(const Mat& m)
Mat::Mat(const Mat& m, const Rect& roi)
Mat::Mat(const Mat& m, const Range* ranges)
Mat::Mat(int ndims, const int* sizes, type)
Mat::Mat(int ndims, const int* sizes, type, const Scalar& s)
Mat::Mat(int ndims, const int* sizes, type, void* data, const size_t step = 0)
```

인수	설명
• int rows, int cols	행렬의 행수와 열수
• int type	행렬 원소의 자료형
• Scalar& s	행렬의 모든 원소 값
• size_t step	행렬의 한 행의 데이터가 차지하는 바이트 수
• void* data	초기화할 행렬 원소 데이터에 대한 포인터
• Size size	행렬의 크기를 Size 객체로 지정
• Rect& roi	관심영역 사각형
• Mat& m	미리 생성된 행렬(기존 Mat 객체로 새 Mat 객체 생성 가능)
• int ndims	행렬의 차원 수(일반적으로 2차원 데이터를 다룸)

생성자

〈표 3.2.1〉 행렬 자료형의 종류

데이터 형	설명	depth 값
CV_8U	uchar(unsigned char)	0
CV_8S	signed char	1
CV_16U	unsigned short int	2
CV_16S	signed short int	3
CV_32S	int	4
CV_32F	float	5
CV_64F	double	6

생성자

예제 3.2.1 Mat 클래스 선언 – mat.cpp

```
01  #include <opencv2/opencv.hpp>
02  using namespace cv;
03  using namespace std;
04  int main()
05  {
06      float data[] = {
07          1.2f, 2.3f, 3.2f,
08          4.5f, 5.f, 6.5f,
09      };
10      // Mat 객체 선언 방법
11      Mat m1(2, 3, CV_8U);
12      Mat m2(2, 3, CV_8U, Scalar(300));          // uchar형 → 255로 저장
13      Mat m3(2, 3, CV_16S, Scalar(300));         // short형 → 300으로 저장
14      Mat m4(2, 3, CV_32F, data);                // 배열 원소로 초기화
15
16      // Size_ 객체로 Mat 객체 선언 방법
17      Size sz(2, 3);
18      Mat m5(Size(2, 3), CV_64F);
19      Mat m6(sz, CV_32F, data);                  // Size_ 객체로 초기화
20
```

2행, 3열 uchar형 행렬 선언- 채널미지정시 1채널

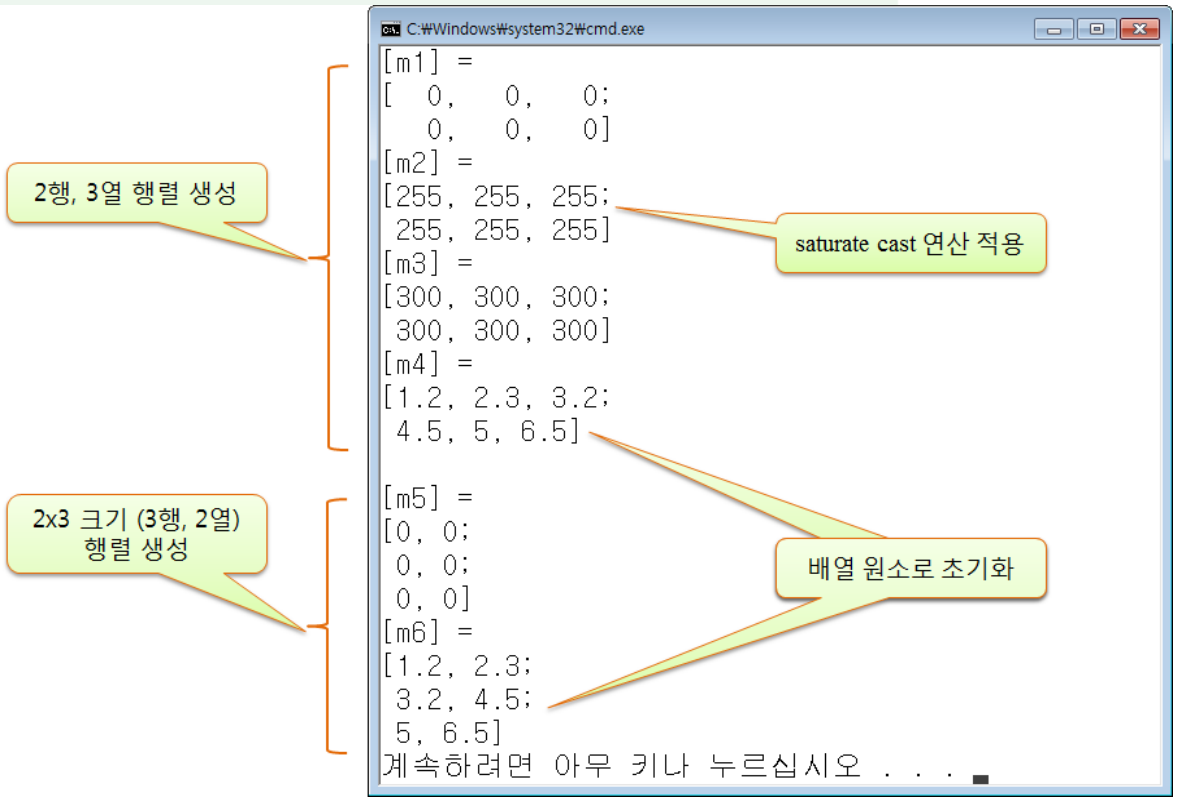
초기값 지정 – uchar형이라 255이상은 255로 저장

float 형 행렬 지정

2x3 크기 행렬 선언 – 3행, 2열

생성자

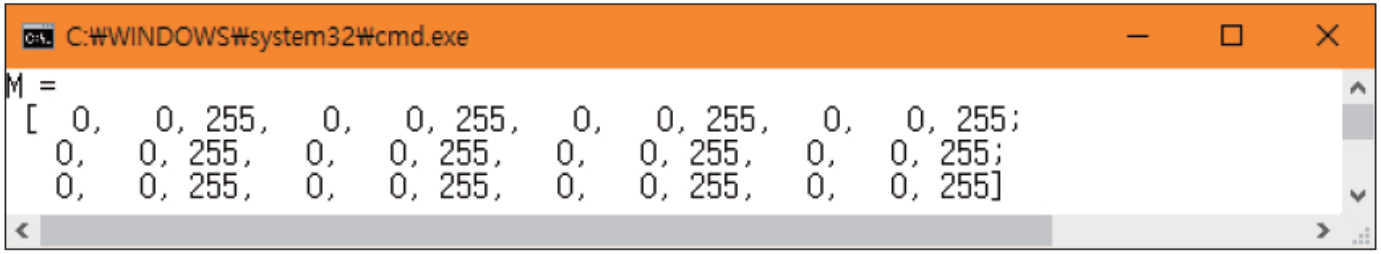
```
20
21      cout << "[m1] =" << endl << m1 << endl;
22      cout << "[m2] =" << endl << m2 << endl;
23      cout << "[m3] =" << endl << m3 << endl;
24      cout << "[m4] =" << endl << m4 << endl << endl;
25      cout << "[m5] =" << endl << m5 << endl;
26      cout << "[m6] =" << endl << m6 << endl;
27      return 0;
28  }
```



생성자

◆ 예제

```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace cv;
using namespace std;

int main()
{
    Mat M(3, 4, CV_8UC3, Scalar(0, 0, 255));
    cout << "M = " << endl << " " << M << endl << endl;
    return 0;
}
```



생성자

◆ 예제

```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace cv;
using namespace std;

int main()
{
    Mat M(600, 800, CV_8UC3, Scalar(0, 255, 0));
    if (M.empty()) { cout << "영상을 읽을 수 없음" << endl;  return -1; }
    imshow("img", M);

    waitKey(0);
    return 0;
}
```

화소 데이터 저장

- 그레이스케일 영상

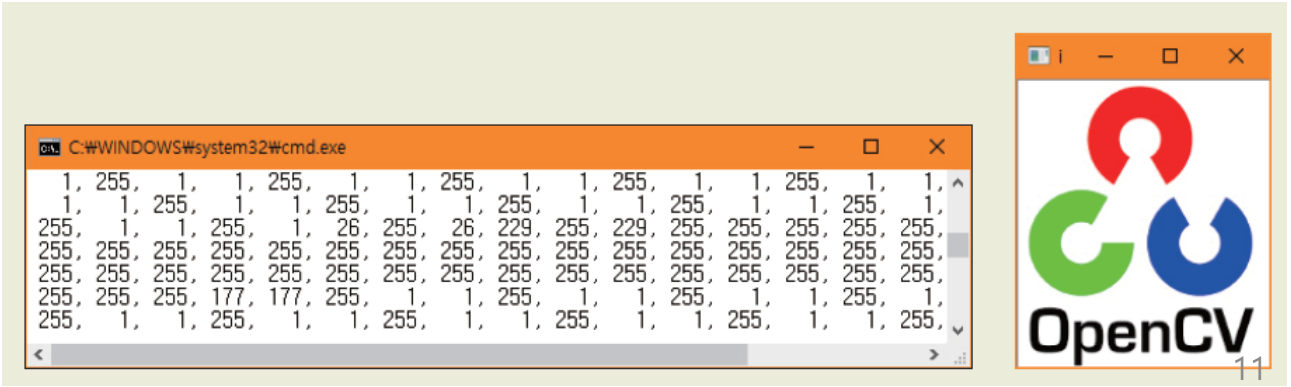
	0열	1열	...	m−1열
0행	(0, 0)	(0, 1)	...	(0, m−1)
1행	(1, 0)	(1, 1)	...	(1, m−1)
...	...	...	...	...
n-1행	(n−1, 0)	(n−1, 1)	...	(n−1, m−1)

- 컬러 영상

화소 데이터 저장

◆ 화소값 출력하기

```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace cv;
using namespace std;

int main()
{
    Mat img = imread("d:/opencv.png");
    if (img.empty()) { cout << "영상을 읽을 수 없음" << endl;  return -1; }
    imshow("img", img);

    cout << img << endl;
    waitKey(0);
    return 0;
}
```



초기화 함수

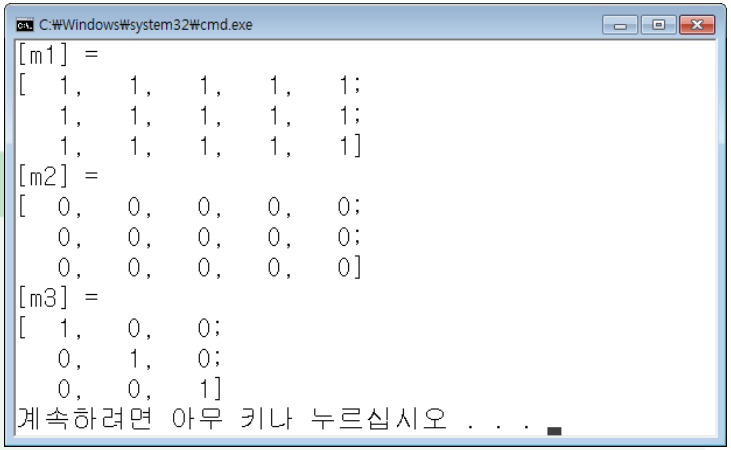
- 단위행렬이나 1로 구성된 행렬 등의 특수한 행렬을 생성하는 함수들

멤버 메서드의 반환 자료형과 인수 구조	

`static MatExpr Mat::ones(int rows, int cols, int type)`
`static MatExpr Mat::ones(Size size, int type)`
`static MatExpr Mat::eye(int rows, int cols, int type)`
`static MatExpr Mat::eye(Size size, int type)`
 	
`static MatExpr Mat::zeros(int rows, int cols, int type)`
`static MatExpr Mat::zeros(Size size, int type)`
`static MatExpr Mat::zeros(int ndims, const int* sz, int type)`	

함수 및 인수	설명
`ones()`	행렬의 모든 원소 1인 행렬을 반환한다.
• `int rows, int cols` • `int type` • `Size size`	행렬의 행의 개수와 열의 개수 행렬 원소의 데이터 타입 행렬의 크기(Size 형)
`eye()`	지정된 크기와 타입의 단위 행렬을 반환한다.
• `int rows, int cols` • `int type` • `Size size`	행렬의 행의 개수와 열의 개수 행렬 원소의 데이터 타입 행렬의 크기(Size 형)
`zeros()`	행렬의 원소를 0으로 초기화 한다.
• `int rows, int cols` • `int type` • `int ndims` • `int* sz`	행렬의 행과 열의 개수 행렬 원소의 데이터 타입 행렬의 차원 수 행렬의 크기를 나타내는 정수 배열

초기화 함수



예제 3.2.2 Mat 클래스 초기화 – mat_init1.cpp

```
01  #include <opencv2/opencv.hpp>
02  using namespace cv;
03  using namespace std;
04  int main()
05  {
06      Mat m1 = Mat::ones(3, 5, CV_8UC1);
07      Mat m2 = Mat::zeros(3, 5, CV_8UC1);
08      Mat m3 = Mat::eye(3, 3, CV_8UC1);
09
10      cout << "[m1] =" << endl << m1 << endl;
11      cout << "[m2] =" << endl << m2 << endl;
12      cout << "[m3] =" << endl << m3 << endl;
13      return 0;
14  }
```

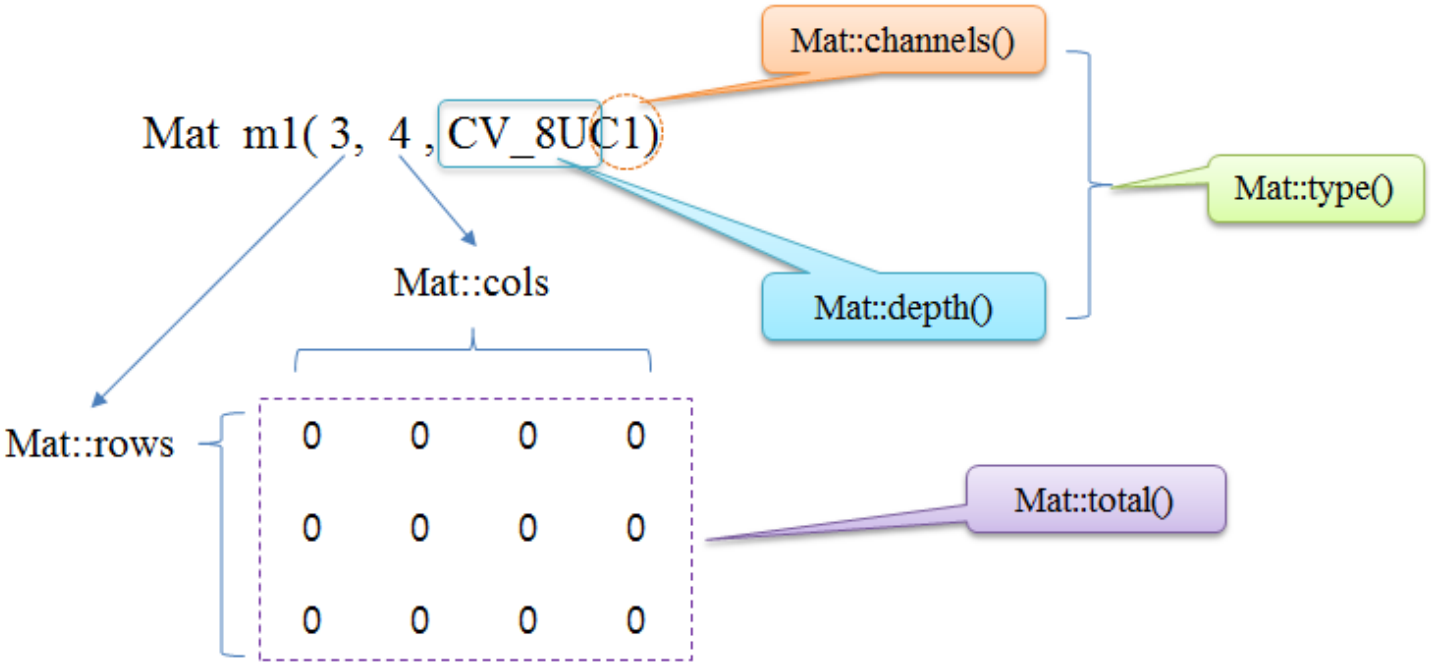
모든 원소가 1로 초기화

모든 원소가 0으로 초기화

단위 행렬 생성

속성 함수

- Mat 객체 선언 시 정보
  - 해당 객체의 멤버 변수들에 저장
  - Mat 클래스의 내부 메서드를 통해 정보 확인 가능



속성 함수

〈표 3.2.2〉 Mat 클래스의 멤버변수 및 멤버(내부) 메서드

항목		설명
멤버 변수	Mat::dims	차원 수
	Mat::rows	행의 개수
	Mat::cols	열의 개수
	Mat::data	행렬 원소 데이터에 대한 포인터
	Mat::step	행렬의 한 행이 차지하는 바이트 수
멤버 메서드	Mat::channels()	행렬의 채널 수 반환
	Mat::depth()	행렬의 깊이(행렬의 자료형)값 반환
	Mat::elemSize()	행렬의 한 원소에 대한 바이트 크기 반환
	Mat::elemSize1()	행렬의 한 원소의 한 채널에 대한 바이트 크기 반환
	Mat::empty()	행렬 원소가 비어있는지 여부 반환
	Mat::isSubmatrix()	참조 행렬인지 여부 반환
	Mat::size()	행렬의 크기를 Size형으로 반환
	Mat::step1()	step을 elemSize1()로 나누어서 정규화된 step 반환
	Mat::total()	행렬 원소의 전체 개수 반환
	Mat::type()	행렬의 데이터 타입(자료형 + 채널 수) 반환 자료형으로 상위 3비트 + 채널 수로 하위 3비트

속성 함수

예제 3.2.5 Mat 클래스 속성 보기 – mat_attr.cpp

```
01  #include <opencv2/opencv.hpp>
02  using namespace cv;
03  using namespace std;
04  int main()
05  {
06      Mat m1(4, 3, CV_32FC3);
07      cout << "차원 수 = " << m1.dims << endl;
08      cout << "행 개수 = " << m1.rows << endl;
09      cout << "열 개수 = " << m1.cols << endl;
10      cout << "행렬 크기 = " << m1.size() << endl << endl;
11
12      cout << "전체 원소 개수 = " << m1.total() << endl;
13      cout << "한 원소의 크기 = " << m1.elemSize() << endl;
14      cout << "채널당 한 원소의 크기 = " << m1.elemSize1() << endl << endl;
15
16      cout << "타입 = " << m1.type() << endl;
17      cout << "타입(채널 수|깊이) = " << ((m1.channels() - 1) << 3) + m1.depth() <<endl;
18      cout << "깊이 = " << m1.depth() << endl;
19      cout << "채널 = " << m1.channels() << endl << endl;
20
21      cout << "step = " << m1.step << endl;
22      cout << "step1() = " << m1.step1() << endl;
23      return 0;
24  }
```

4행, 3열 3채널 float 행렬

```
C:₩Windows₩system32₩cmd.exe
차원 수 =2
행 개수 =4
열 개수 =3
행렬 크기 = [3 x 4]

전체 원소 개수 = 12
한 원소의 크기 = 12
채널당 한 원소의 크기 = 4

타입 = 21
타입(채널수 | 깊이) = 21
깊이 = 5
채널 = 3

step = 36
step1() = 9
계속하려면 아무 키나 누르십시오 . . .
```

속성 함수

```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace cv;
using namespace std;

int main()
{
    Mat img = imread("d:/lenna.jpg");
    if (img.empty()) { cout << "영상을 읽을 수 없음" << endl; return -1; }
    imshow("img", img);

    cout << "행의 수 = " << img.rows << endl;
    cout << "열의 수 = " << img.cols << endl;
    cout << "행렬의 크기 = " << img.size() << endl;
    cout << "전체 화소 개수 = " << img.total() << endl;
    cout << "한 화소 크기 = " << img.elemSize() << endl;
    cout << "타입 = " << img.type() << endl;
    cout << "채널 = " << img.channels() << endl;
    waitKey(0);
    return 0;
}
```

◆ Mat 객체의 속성 출력

할당(=) 연산자

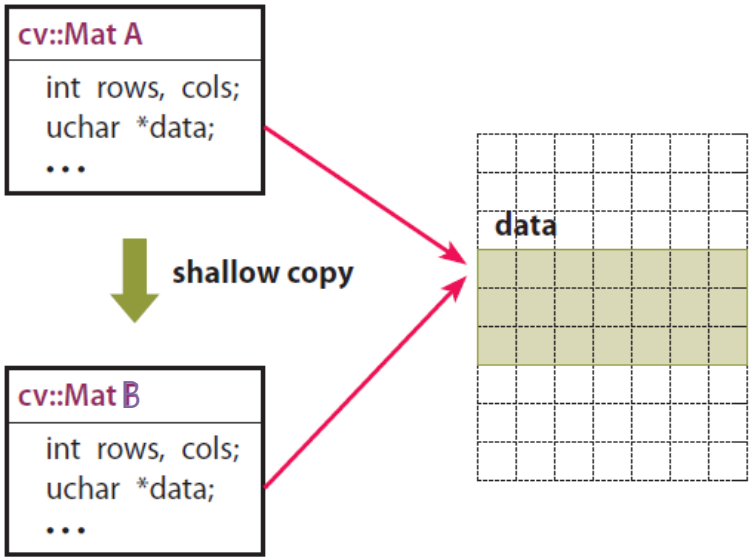
〈표 3.2.3〉 Mat의 할당(=) 연산자 종류

오른쪽 항의 데이터 타입	예문	설명
스칼라값	m1 = 100	행렬의 모든 원소를 지정된 스칼라값으로 변경하고자 할 때 사용한다. 마스크 행렬이 없는 MatsetTo()와 같은 역할을 한다.
행렬 수식	m1 = m2 + m3 m1 = m3 - 6	수식의 결과가 m1 행렬에 복사된다.
행렬	m1 = m2	m2 행렬이 m1 행렬에 복사되는 것이 아니라 m2 행렬을 m1 행렬이 공유한다. 따라서 m2 행렬의 원소가 변경되면, m1 행렬의 원소도 같이 변경된다.

할당(=) 연산자

- 얕은 복사 (Shallow Copy)

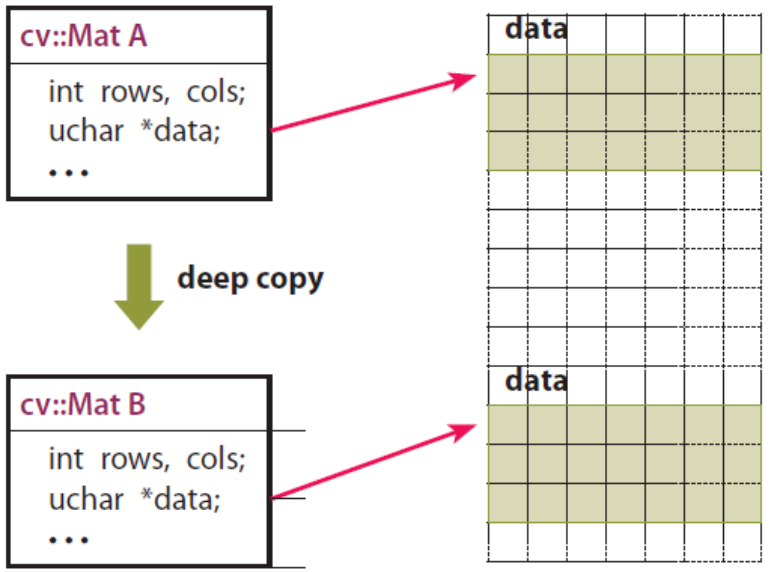
```
Mat A;                          // 여기서는 헤더만 생성된다.
A = imread("d:/dog.jpg", IMREAD_COLOR); // 여기서 메모리가 할당된다.

Mat B = A;                      // 대입 연산자, 얕은 복사가 수행된다.
```



할당(=) 연산자

- 깊은 복사 (Deep Copy)

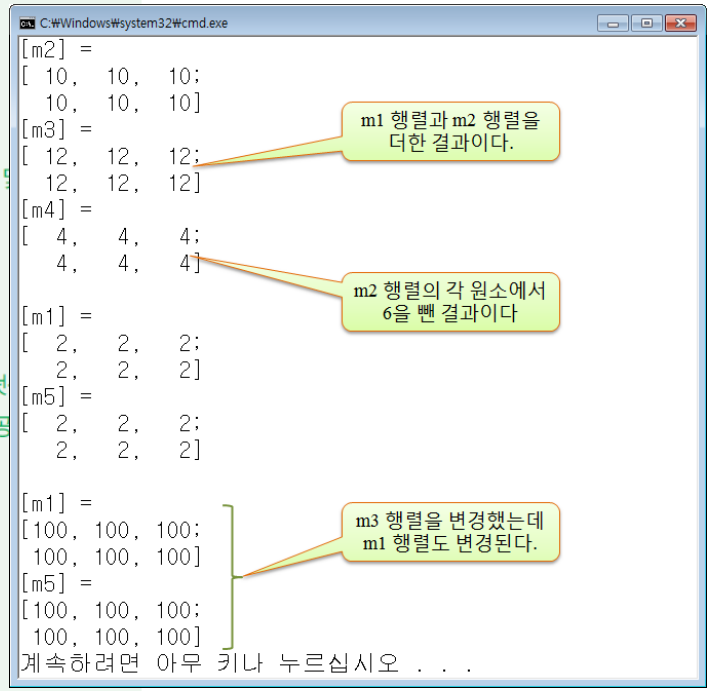
```
Mat A;                                  // 여기서는 헤더만 생성된다.
A = imread("d:/dog.jpg", IMREAD_COLOR); // 여기서 메모리가 할당된다.

Mat B = A.clone();
```



할당(=) 연산자

예제 3.2.6 Mat 객체 연산 – mat_op.cpp

```
01  #include <opencv2/opencv.hpp>
02  using namespace cv;
03  using namespace std;
04  int main()
05  {
06      Mat m1(2, 3, CV_8U, 2);                          // 2행, 3열 행렬 선언 및
07      Mat m2(2, 3, CV_8U, Scalar(10));
08
09      // 행렬 연산
10      Mat m3 = m1 + m2;                                // 원소 간 덧셈
11      Mat m4 = m2 - 6;                                 // 원소와 스칼라값의 덧
12      Mat m5 = m1;                                     // m5 행렬이 m1 행렬 공
13
14      cout << "[m2] =" << endl << m2 << endl;
15      cout << "[m3] =" << endl << m3 << endl;
16      cout << "[m4] =" << endl << m4 << endl << endl;
17
18      // 공유 행렬 처리
19      cout << "[m1] =" << endl << m1 << endl;
20      cout << "[m5] =" << endl << m5 << endl << endl;
21      m5 = 100;
22      cout << "[m1] =" << endl << m1 << endl;
23      cout << "[m5] =" << endl << m5 << endl ;
24      return 0;
25  }
```

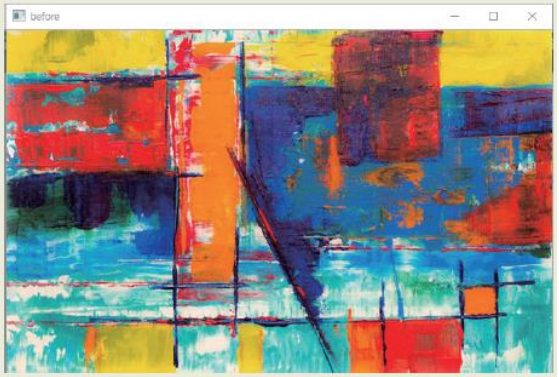


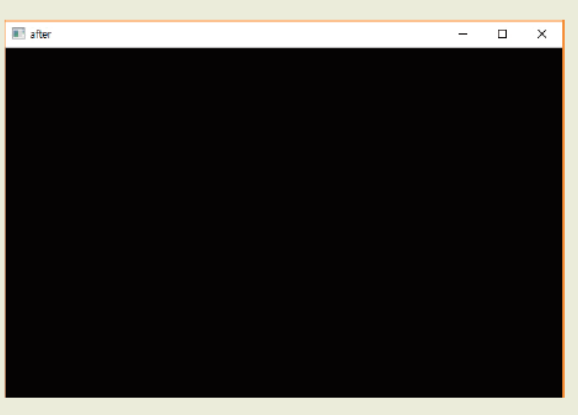
행렬의 모든 원소 100으로 변경.
m5 행렬이 m1 행렬을 공유하기에
m1 행렬의 원소도 같이 변경

할당(=) 연산자

- 예제 (얕은 복사)

```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace cv;
using namespace std;

void sub(Mat img)
{
    img = Scalar(0, 0, 0);      // 영상의 모든 화소를 0으로 만든다.
}
int main()
{
    Mat A;
    A = imread("d:/drawing.jpg", IMREAD_COLOR);

    imshow("before", A);        // 함수 호출 전에 영상을 표시한다.
    sub(A);                     // 함수에 영상을 전달한다.
    imshow("after", A);         // 함수 호출 후에 영상을 표시한다.

    waitKey();
    return 0;
}
```





크기 및 형태 변경

내부 메서드의 반환 자료형과 인수 구조	

`void Mat::resize(size_t sz);`
`void Mat::resize(size_t sz, const Scalar& s)`
`Mat Mat::reshape(int cn, int rows = 0);`
`void Mat::create(int rows, int cols, int type)`
`void Mat::create(Size size, int type)`
`void Mat::create(int ndims, const int* sizes, int type)`	

함수 및 인수	설명
`void resize()`	행의 개수를 기준으로 기존 행렬의 크기를 변경한다. 기존 행렬의 행의 개수 보다 sz가 작으면 하단 행을 제거하고, 크면 기존 행렬 하단에 행을 추가한다.
• `size_t sz` • `Scalar& s`	변경될 행의 개수 추가되는 행의 원소에 할당하는 스칼라값
`Mat reshape()`	행렬의 전체 원소 개수는 바뀌지 않으면서, 행렬의 모양을 변경하여 새 행렬을 반환한다. 기존 행렬과 변경된 행렬의 전체 원소 개수(채널 수×행수×열수)가 일치하지 않으면 에러가 발생한다.
• `int cn` • `int rows`	변경될 채널 개수 변경될 행의 개수
`void create()`	기존에 존재하는 행렬의 차원, 행, 열, 자료형을 변경하여 다시 생성한다. 기존 행렬과 크기와 자료형이 다르면 기존 메모리를 해제하고 새로운 데이터를 생성한다.
• `int rows, int cols` • `int type` • `int ndims` • `int* sizes`	행렬의 행과 열의 개수 행렬 원소의 데이터 타입(자료형 + 채널 수) 행렬의 차원 수 행렬의 크기를 나타내는 정수 배열

크기 변경

- void Mat::resize (Size)

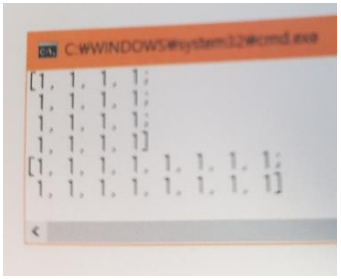
```
#include "opencv2/opencv.hpp"
using namespace cv;
using namespace std;

int main()
{
    Mat LoadedImage = imread("d:/Lenna.jpg", IMREAD_COLOR);
    imshow("Original Image", LoadedImage);

    LoadedImage.resize(Size(100, 100));
    imshow("New Image", LoadedImage);
    waitKey(0);
    return 0;
}
```

영상의 크기를 100X100으로 만든다.

형태 변경

- Mat Mat::reshape (int channels, int rows=0)

```
#include "opencv2/opencv.hpp"
using namespace cv;
using namespace std;

int main()
{
    Mat m = Mat::ones(4, 4, CV_32FC1);
    cout << m << endl;

    m = m.reshape(0, 2);
    cout << m << endl;

    return 0;
}
```



복사 및 자료형 변환

- 영상처리 시 원본 행렬 복사 빈번히 발생

멤버 메서드의 반환 자료형과 인수 구조	
`Mat Mat::clone()`	
`void Mat::copyTo(Mat &mat)`	
`void Mat::copyTo(Mat &mat, Mat mask)`	
`void Mat::converTo(Mat &mat, int type, double alpha = 1, double beta = 0)`	
함수 및 인수	**설명**
`Mat clone()`	행렬 데이터와 같은 값을 복사해서 새로운 행렬을 반환한다
`void copyTo()`	행렬 데이터를 인자로 입력된 mat 행렬에 복사한다.
• `Mat mat`	복사될 목적 행렬
• `Mat mask`	연산 마스크(mask 행렬의 원소가 0이 아닌 위치만 복사가 수행)
`void converTo()`	행렬 원소의 데이터 타입을 변경하여 인수로 입력된 mat 행렬에 반환한다.
• `Mat mat`	데이터 타입이 변경될 목적 행렬
• `int type`	변경하고자 하는 데이터 타입(예, CV_8U , CV_16S 등)
• `double alpha`	원본 행렬의 원소 값의 배율 지정
• `double beta`	원본 행렬의 원소 값에 대한 이동값(delta)

복사 및 자료형 변환

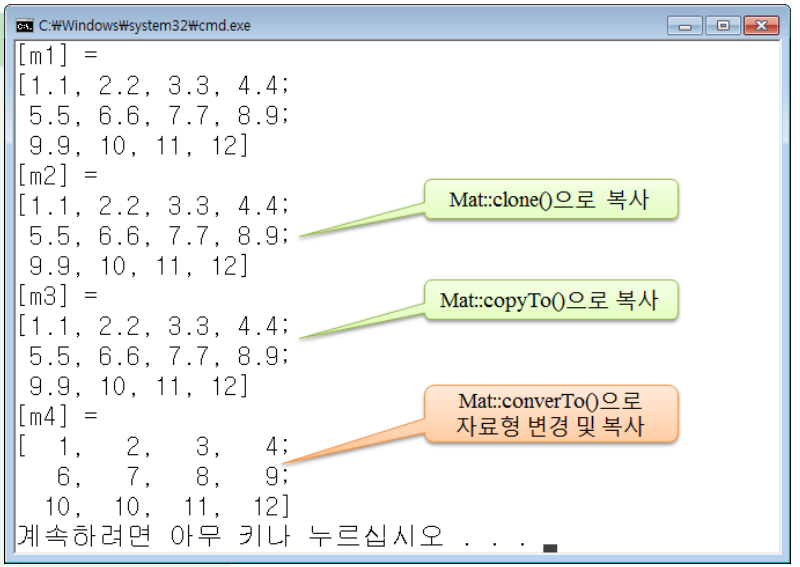
예제 3.2.9 Mat 객체 복사– mat_copy.cpp

```
01  #include <opencv2/opencv.hpp>
02  using namespace cv;
03  using namespace std;
04  int main()
05  {
06      double data[] = {
07          1.1, 2.2, 3.3, 4.4,
08          5.5, 6.6, 7.7, 8.9,
09          9.9, 10 , 11, 12
10      };
11      Mat m1(3, 4, CV_64F, data);
12      Mat m2 = m1.clone();
13      Mat m3, m4 ;
14      m1.copyTo(m3);
15      m1.convertTo(m4, CV_8U);                    // 형변환을 통한 복사의 한 방법
16
17      cout << "m1 =\n" << m1 << endl;
18      cout << "m2 =\n" << m2 << endl;
19      cout << "m3 =\n" << m3 << endl;
20      cout << "m4 =\n" << m4 << endl;
21      return 0;
22  }
```

m1행렬 복사후 m2에 저장

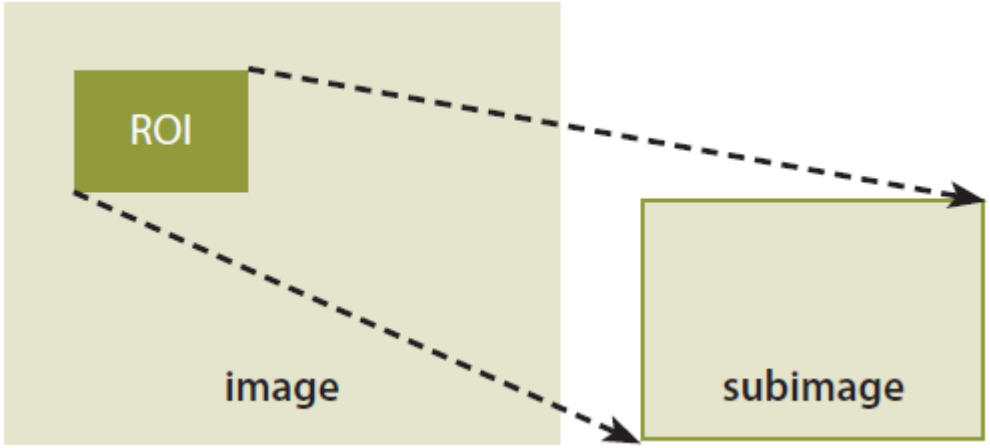
m1행렬 복사후 m3에 저장

double 행렬 → uchar행렬로 형변환 후 m4에 저장



관심 영역 지정

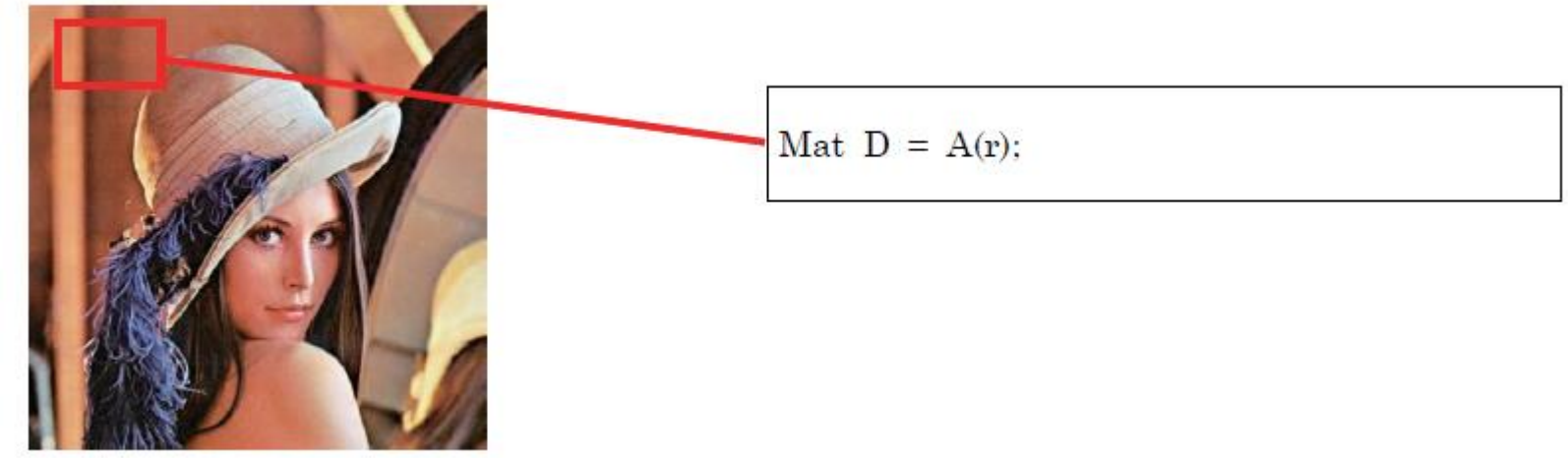
관심영역 (ROI, Region of Interset)



관심 영역 지정

- 사각형 형태의 관심 영역 지정하기

```
A = imread("d:/lenna.jpg", IMREAD_COLOR); // 여기서 메모리가 할당된다.
Rect r(10, 10, 100, 100);
Mat D = A(r);       // 사각형을 사용하여 관심 영역을 지정한다.
```



관심 영역 지정

- 예제

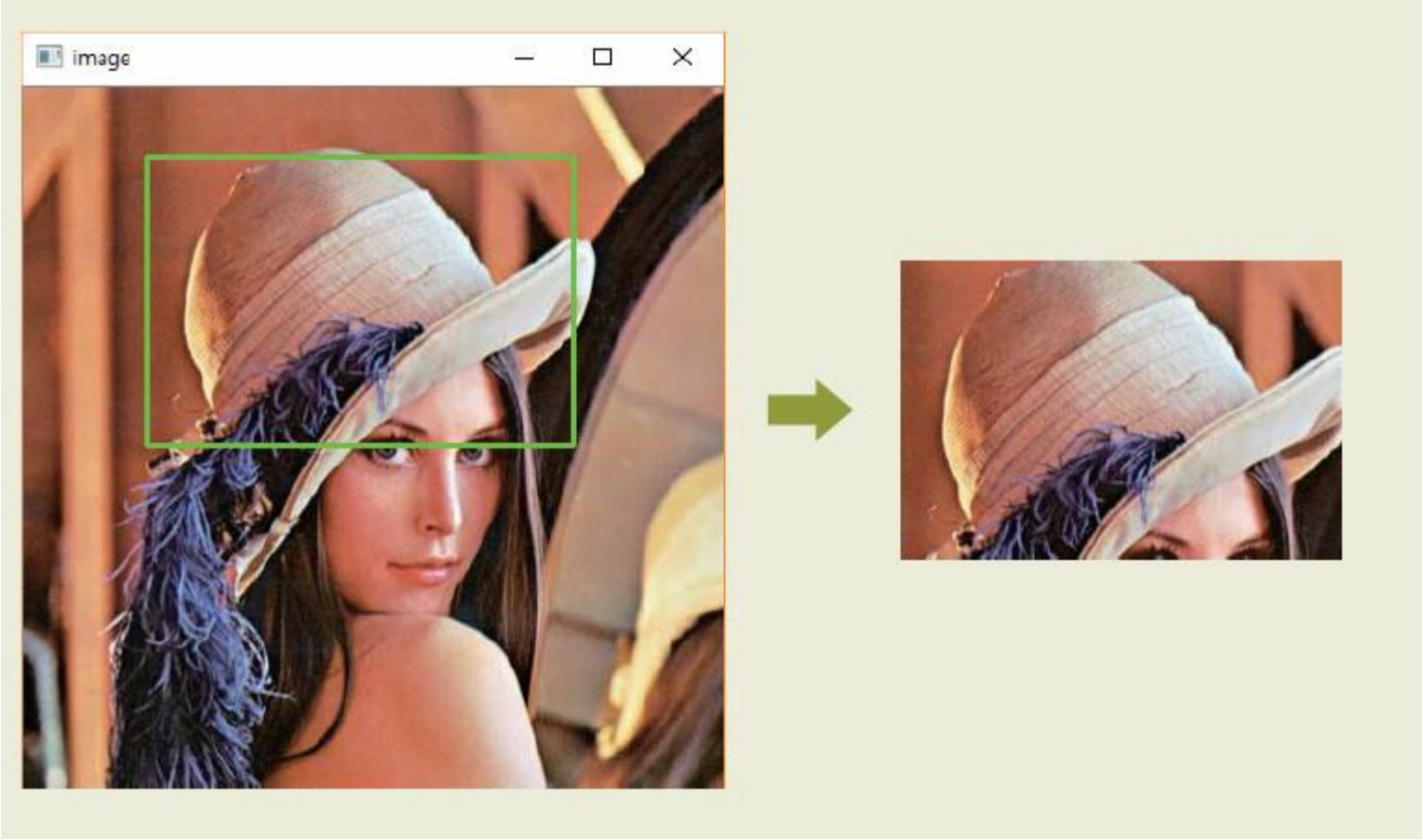
```
...
int main()
{
    Mat A;
    A = imread("d:/lenna.jpg", IMREAD_COLOR);

    Rect r(10, 10, 100, 100);
    Mat D = A(r);              // 사각형을 사용하여 관심 영역을 지정한다.
    D = Scalar(0, 0, 0);       // 관심 영역의 모든 화소가 (0, 0, 0)이 된다.
    imshow("src", A);
    waitKey();
    return 0;
}
```

관심 영역 지정

- 로고 삽입하기

```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace cv;
using namespace std;

int main() {
    Mat A = imread("d:/image2.jpg");
    Mat B = imread("d:/logo.png");
    Mat roi(A, Rect(A.cols - B.cols, A.rows - B.rows, B.cols, B.rows));
    B.copyTo(roi);

    imshow("result", A);
    waitKey(0);
    return 0;
}
```

미니 포토샵 만들기



미니 포토샵 만들기

```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace cv;
using namespace std;

Mat img, roi;
int mx1, my1, mx2, my2;                    // 마우스로 지정한 사각형의 좌표
bool cropping = false;                     // 사각형 선택 중임을 나타내는 플래그 변수

// 마우스 이벤트가 발생하면 호출되는 콜백 함수이다.
void onMouse(int event, int x, int y, int flags, void* param)
{
        if (event == EVENT_LBUTTONDOWN) {// 마우스의 왼쪽 버튼을 누르면
                mx1 = x;                                    // 사각형의 좌측 상단 좌표 저장
                my1 = y;
                cropping = true;
        }
        else if (event == EVENT_LBUTTONUP) {               // 마우스의 왼쪽 버튼에서 손을 떼면
                mx2 = x;                                    // 사각형의 우측 하단 좌표 저장
                my2 = y;
                cropping = false;
                rectangle(img, Rect(mx1, my1, mx2 - mx1, my2 - my1), Scalar(0, 255, 0), 2);
                imshow("image", img);
        }
}
```

미니 포토샵 만들기

```
int main() {
        img = imread("d:/lenna.jpg");
        imshow("image", img);
        Mat clone = img.clone();                    // 복사본을 만들어둔다.

        setMouseCallback("image", onMouse);

        while (1) {
                int key = waitKey(100);
                if (key == 'q') break;      // 사용자가 'q'를 누르면 종료
                else if (key == 'c') {      // 사용자가 'c'를 누르면 관심영역을 파일로 저장
                        roi = clone(Rect(mx1, my1, mx2 - mx1, my2 - my1));
                        imwrite("d:/result.jpg", roi);
                }
        }
        return 0;
}
```

HW

1

다음은 행렬을 참조하여 부분 행렬을 만드는 예제 소스이다. 이 예제의 실행 결과를 적으시오.

```
int main()
{
    Range  r1(2, 3), r2(3, 5);

    int data[] = {
        10, 11, 12, 13, 14, 15, 16,
        20, 21, 22, 23, 24, 25, 26,
        30, 31, 32, 33, 34, 35, 36,
        40, 41, 42, 43, 44, 45, 46,
    };
    Mat m1(5, 7, CV_32S, data);

    cout << m1(r1, r2) ;
}
```

2

다음과 같이 출력하는 프로그램을 작성하시오.

1) 행렬은 int형이며, 10x15 크기이다.
2) 행렬의 출력 결과는 다음과 같다.
3) 메인 함수내의 코드 라인수는 10행 이내로 작성한다.