



Chapter 06. 배열과 포인터

목차

1. 배열의 이해
2. 1차원 배열과 포인터
3. 2차원 포인터

학습목표

- 동일한 자료형의 데이터 집합인 배열에 대해서 학습한다.
- 1차원 배열과 포인터의 관계를 학습한다.
- 2차원 포인터의 다양한 형태를 학습한다.

01 배열의 이해

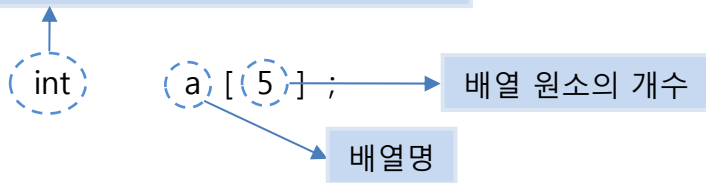
■ 1차원 배열

- 배열에는 데이터를 여러 개 저장할 수 있으며, 배열에 저장된 각각의 값을 배열의 원소(element)라고 한다.

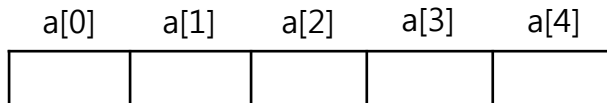
자료형 배열명[원소의 개수];

배열 기본 형식

각 원소에 저장할 값에 대한 자료형



[그림 6-1] 배열 선언 구조의 예



[그림 6-2] 배열의 메모리 할당

배열명[0] ~ 배열명[n-1]

배열의 원소 개수 선언 형식

01 배열의 이해

■ 1차원 배열의 초기화

- 배열의 개수와 초기값의 개수가 동일한 경우다.

```
int a[5] = {10, 20, 30, 40, 50};
```

- 배열에 초기값이 지정되어 있을 경우에는 배열의 개수를 생략할 수 있다.

```
int a[ ] = {10, 20, 30};
```

- 배열의 개수보다 초기값이 적을 경우에는 나머지 영역에 0을 채운다.

```
int a[5] = {10, 20, 30};
```

- 배열의 개수보다 초기값을 많이 주었을 경우에는 에러가 발생한다.

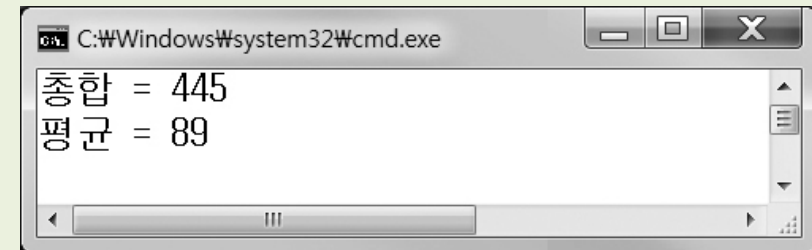
```
int a[5] = {10, 20, 30, 40, 50, 60, 70} //에러
```

- 배열의 초기값을 주지 않은 상태에서는 배열의 원소 개수를 생략할 수 없다.

```
int a[ ]; //에러
```

예제 6-2. 배열을 사용해서 총합과 평균 구하기(06_02.cpp)

```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05     int a[5]={85, 90, 75, 100, 95};
06     int tot=0;
07     double avg;
08     int i;
09
10     for(i=0; i<5; i++)
11         tot+=a[i];
12
13     avg = (double)tot/5.0;
14
15     cout << "총합 = " << tot << "\n";
16     cout << "평균 = " << avg << "\n";
17 }
```



01 배열의 이해

■ 2차원 배열

- 2차원 배열은 2차원 배열용 자료형이 따로 있는 것이 아니라 1차원 배열을 선언하는 형식에 행과 열의 개수만 추가로 지정하면 된다.

자료형 배열명[행의 개수][열의 개수];

2차원 배열 선언 기본 형식

```
int a[3][4];
```

	0열	1열	2열	3열
0열	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1열	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2열	a[2][0]	a[2][1]	a[2][2]	a[2][3]

a[1][2] = 10;

[그림 6-3] 2차원 배열의 구조

01 배열의 이해

■ 2차원 배열의 초기화

```
int a[3][4] = { {90, 85, 95, 100}, // 0번째 행에 대한 초기화
               {75, 95, 80, 90},   // 1번째 행에 대한 초기화
               {90, 80, 70, 60}    // 2번째 행에 대한 초기화
               };
```

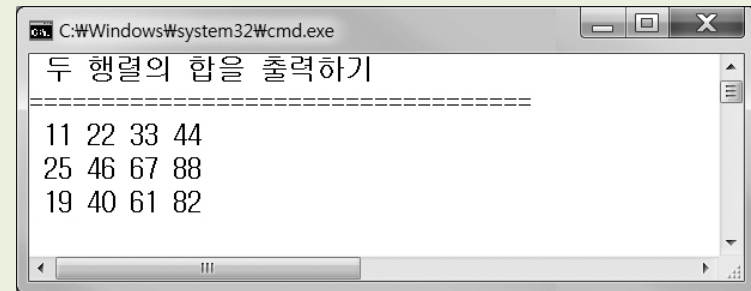
- 2차원 배열의 초기값에 중괄호를 중첩해 사용한다.

■ 이중 for문을 이용한 2차원 배열의 출력

int a[3][4];	=	<pre>cout << "Wt" << a[0][0]; cout << "Wt" << a[0][1]; cout << "Wt" << a[0][2]; cout << "Wt" << a[0][3]; cout << "Wn"; cout << "Wt" << a[1][0]; cout << "Wt" << a[1][1]; : : cout << "Wt" << a[2][2]; cout << "Wt" << a[2][3];</pre>	=	<pre>for(r= 0; r<3; r++) { for(c= 0; c<4; c++) cout<<"Wt"<<a[r][c]; cout<<"Wn"; }</pre>
--------------	---	--	---	---

예제 6-5. 두 행렬의 합 구하기(06_05.cpp)

```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05
06     int a[3][4]={ {10, 20, 30, 40}, {20, 40 ,60 ,80}, {10, 30, 50, 70} };
07     int b[3][4]={ { 1, 2, 3, 4}, { 5, 6, 7, 8}, { 9, 10 ,11 ,12} };
08     int c[3][4];
09     int row, col;
10     for(row = 0; row < 3; row ++ )
11         for(col = 0; col < 4; col ++ )
12             c[row][col] = a[row][col]+ b[row][col];
13
14     cout<<" 두 행렬의 합을 출력하기";
15     cout<<"\n===== \n";
16     for(row = 0; row < 3; row ++ ){
17         for(col = 0; col < 4 ; col ++ )
18             cout<<" "<<c[row][col];
19         cout<<"\n";
20     }
21 }
```



02 1차원 배열과 포인터

■ 배열 원소의 주소값

- 배열 원소의 주소값은 배열에 첨자를 지정한 원소에 &을 붙이면 된다.

```
int a[10];  
배열 원소의 주소값 == &a[2]
```

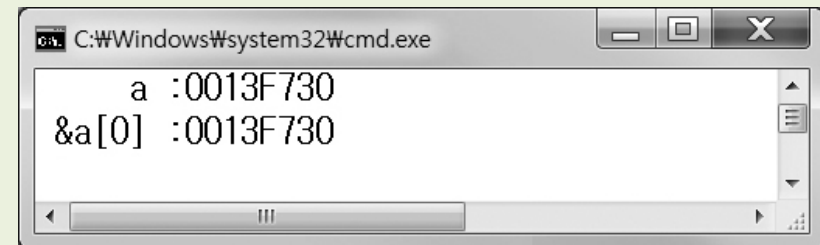
■ 배열명과 포인터

- 배열명만 기술하면 배열의 시작 주소값, 즉 포인터로 해석한다.

```
int a[10];  
배열의 시작 주소값 == a == &a[0]
```

예제 6-8. 배열명의 값을 출력하기(06_08.cpp)

```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05     int a[5] = {10, 20, 30, 40, 50};
06
07     cout << "    a : " << a << "\n";
08     cout << " &a[0] : " << &a[0] << "\n";
09     cout << " &a[1] : " << &a[1] << "\n";
10 }
```



02 1차원 배열과 포인터

■ 배열과 포인터 연산

- 배열명은 배열의 시작 주소값을 알려주는 포인터이다.

배열의 시작 주소값 == a == &a[0]

- *a와 같이 배열명 앞에 * 연산자를 붙이면 그 주소를 찾아가 해당 기억공간에 저장된 값을 알려준다.
즉, 배열명에 * 연산자를 붙이면 배열의 시작 주소에 있는 첫 번째 원소값을 알려준다.

*a == *(&a[0]) == a[0]

- 배열의 첨자가 i인 원소의 주소는 a[i]앞에 & 연산자를 붙일 수도 있지만 a+i와 같이 배열명에 i를 더해 구할 수도 있다.

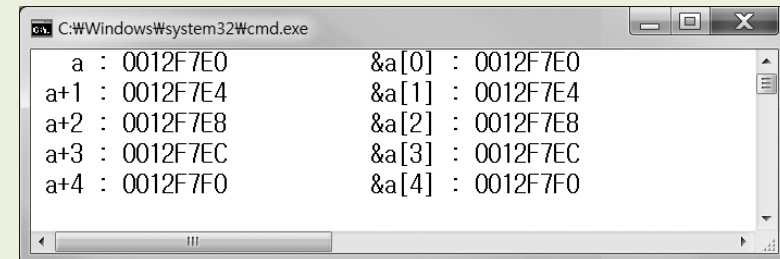
a+i == &a[i];

[표 6-1] 포인터에 사용할 수 있는 산술 연산자

연산자	의미	결과
+	다음에 나올 원소의 주소값을 알려준다.	주소(포인터)
-	이전 원소의 주소값을 알려준다.	주소(포인터)

예제 6-9. 배열명과 + 연산자의 관계 알아보기(06_09.cpp)

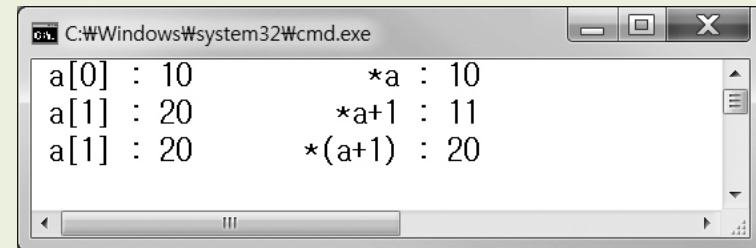
```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05     int a[5] = {10, 20, 30, 40, 50};
06     cout<<" a : "<<a <<" \t &a[0] : "<<&a[0]<<"\n" ;
07     cout<<" a+1 : "<<a+1 <<" \t &a[1] : "<<&a[1]<<"\n" ;
08     cout<<" a+2 : "<<a+2 <<" \t &a[2] : "<<&a[2]<<"\n" ;
09     cout<<" a+3 : "<<a+3 <<" \t &a[3] : "<<&a[3]<<"\n" ;
10     cout<<" a+4 : "<<a+4 <<" \t &a[4] : "<<&a[4]<<"\n" ;
11 }
```



```
C:\Windows\system32\cmd.exe
a : 0012F7E0      &a[0] : 0012F7E0
a+1 : 0012F7E4    &a[1] : 0012F7E4
a+2 : 0012F7E8    &a[2] : 0012F7E8
a+3 : 0012F7EC    &a[3] : 0012F7EC
a+4 : 0012F7F0    &a[4] : 0012F7F0
```

예제 6-10. 포인터를 사용해서 배열의 각 원소값 출력하기(06_10.cpp)

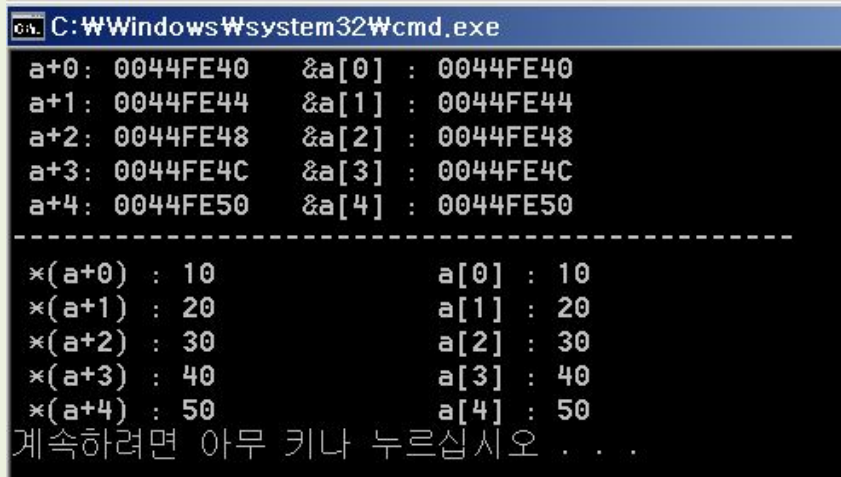
```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05     int a[5] = {10, 20, 30, 40, 50};
06
07     cout<<" a[0] : "<<a[0]<<" Вт *a : "<< *a<<"\n";
08     cout<<" a[1] : "<<a[1]<<" Вт *a+1 : "<< *a+1<<"\n";
09     cout<<" a[1] : "<<a[1]<<" Вт *(a+1) : "<<*(a+1)<<"\n";
10 }
```



```
C:\Windows\system32\cmd.exe
a[0] : 10 *a : 10
a[1] : 20 *a+1 : 11
a[1] : 20 *(a+1) : 20
```

예제 6-11. 포인터 형태로 배열의 원소값 출력(06_11.cpp)

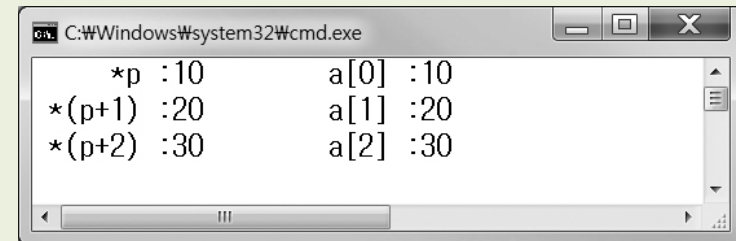
```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05     int a[5] = {10, 20, 30, 40, 50};
06     int i;
07     for(i=0;i<5;i++)
08         cout << " a+" << i << ": " << a + i << "Wt &a[" << i << "] : " << &a[i] << "Wn";
09     cout<<"-----Wn";
10     for(i=0;i<5;i++)
11         cout << " *(a+" << i << ") : " << *(a + i) << "WtWt a[" << i << "] : " << a[i] << "Wn";
12 }
```



```
C:\Windows\system32\cmd.exe
a+0: 0044FE40 &a[0] : 0044FE40
a+1: 0044FE44 &a[1] : 0044FE44
a+2: 0044FE48 &a[2] : 0044FE48
a+3: 0044FE4C &a[3] : 0044FE4C
a+4: 0044FE50 &a[4] : 0044FE50
-----
*(a+0) : 10 a[0] : 10
*(a+1) : 20 a[1] : 20
*(a+2) : 30 a[2] : 30
*(a+3) : 40 a[3] : 40
*(a+4) : 50 a[4] : 50
계속하려면 아무 키나 누르십시오 . . .
```

예제 6-12. 배열명과 포인터 변수의 관계 알아보기(06_12.cpp)

```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05     int a[5] = {10,20,30,40,50};
06     int *p;           // 포인터 변수 선언
07     p = a;            // 포인터 변수 초기화
08
09     cout << "₩n *p :." << *p;
10     cout << "₩t a[0] :." << a[0];
11
12     cout << "₩n *(p+1) :." << *(p+1);
13     cout << "₩t a[1] :." << a[1];
14
15     cout << "₩n *(p+2) :." << *(p+2);
16     cout << "₩t a[2] :." << a[2] << "₩n";
17 }
```



```
C:\Windows\system32\cmd.exe
    *p :10      a[0] :10
  *(p+1) :20    a[1] :20
  *(p+2) :30    a[2] :30
```


03 2차원 포인터

① 포인터의 포인터 (2차원 포인터)

- 1차원 포인터의 주소값을 저장하는 포인터의 포인터는 * 기호를 두 번씩 기술해서 선언한다.

자료형 **포인터 변수명;

포인터의 포인터 기본 형식

EX) int **pp; // 이차원 포인터

- 포인터와 관련된 연산자인 & 과 *

```
int a = 10;  
int *p = &a;
```

포인터 변수 p는 변수 a의 값이 저장된 곳의 주소값이다.

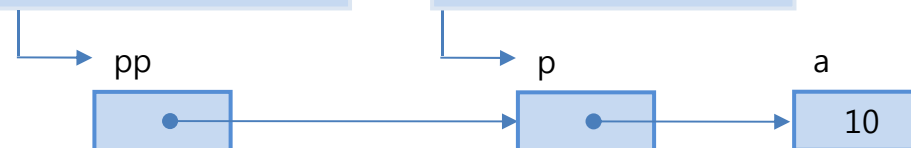


- 포인터의 포인터

```
int **pp = &p;
```

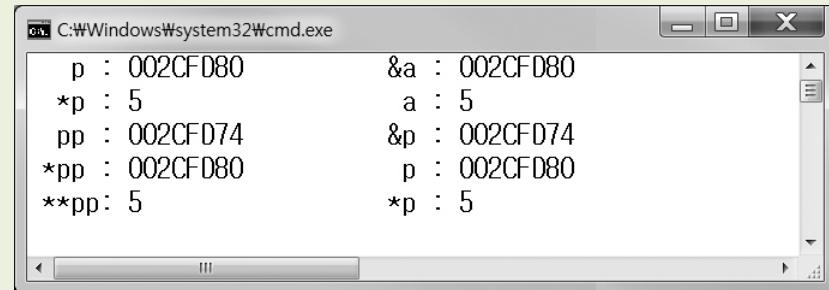
포인터 변수 pp는
포인터 변수 p의 값이
저장된 곳의 주소값이다.

포인터 변수 p는
변수 a의 값이
저장된 곳의 주소값이다.



예제 6-13. 2차원 포인터 사용하기(06_13.cpp)

```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05     int a=5;
06     int *p;
07     int **pp;
08
09     p=&a;
10     pp=&p;
11
12     cout<<" p : "<< p <<" \t &a : "<< &a << endl;
13     cout<<" *p : "<< *p <<" \t \t a : "<< a << endl;
14     cout<<" pp : "<< pp <<" \t &p : "<< &p << endl;
15     cout<<" *pp : "<< *pp <<" \t p : "<< p << endl;
16     cout<<" **pp: "<< **pp <<" \t \t *p : "<< *p << endl;
17 }
```



```
C:\Windows\system32\cmd.exe
p : 002CFD80      &a : 002CFD80
*p : 5           a : 5
pp : 002CFD74     &p : 002CFD74
*pp : 002CFD80    p : 002CFD80
**pp: 5          *p : 5
```

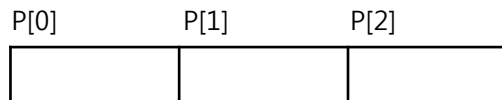
03 2차원 포인터

② 1차원 포인터를 저장하는 포인터 배열

❶ `int *p1, *p2, *p3;`

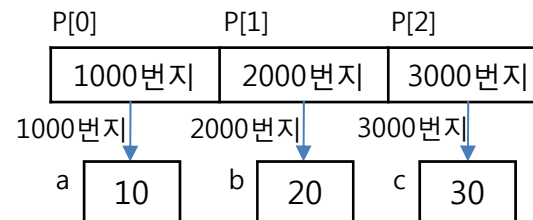
❷ `int *p[3];`

- ❷처럼 작성하면 원소 3개를 저장할 수 있는 배열이 생성되며, 각 원소에는 1차원 포인터를 저장할 수 있다.



- 각 원소에 1차원 포인터를 저장해 보자.

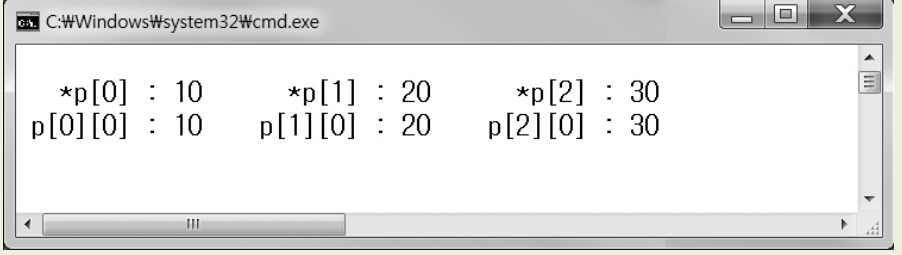
```
int a=10, b= 20, c=30;  
int *p[3]={&a, &b, &c};
```



- 각 원소의 내용은 주소이므로 10, 20, 30을 출력하려면 각 원소 앞에 * 연산자를 기술해야 한다.
즉, `p[0]==&a` 이므로 `*(p[0])==*p[0]==a`가 된다.

예제 6-14. 1차원 포인터를 저장하는 포인터 배열(06_14.cpp)

```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05     int a=10, b= 20, c=30; // 정수형 변수
06     // 포인터 배열에 변수의 주소를 저장해 둔다.
07     int *p[3]={&a, &b, &c};
08
09     // 배열 원소에 * 연산자로 정수값을 얻어온다.
10     cout<<"\n *p[0] : "<< *p[0];
11     cout<<"\t *p[1] : "<< *p[1];
12     cout<<"\t *p[2] : "<< *p[2];
13
14     // * 연산자 대신 [ ]로 정수값을 얻어온다.
15     cout<<"\n p[0][0] : "<< p[0][0];
16     cout<<"\t p[1][0] : "<< p[1][0];
17     cout<<"\t p[2][0] : "<< p[2][0];
18     cout<<"\n";
19 }
```



```
C:\Windows\system32\cmd.exe

*p[0] : 10      *p[1] : 20      *p[2] : 30
p[0][0] : 10    p[1][0] : 20    p[2][0] : 30
```

03 2차원 포인터

- 포인터 배열에 1차원 배열명인 배열의 시작 주소를 저장해 보자.

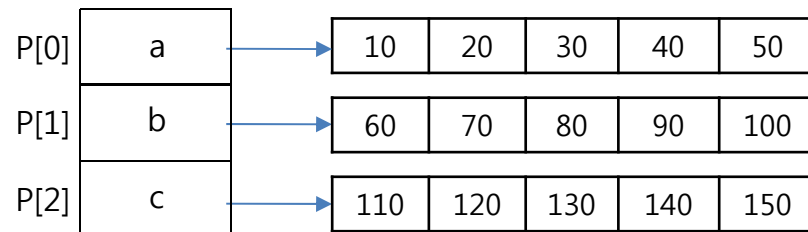
```
Int a[5]={10, 20, 30, 40, 50};
```

```
Int b[5]={60, 70, 80, 90, 100};
```

```
Int c[5]={110, 120, 130, 140, 150};
```

```
Int *p[3]={a, b, c};
```

- 배열명 자체가 포인터이기 때문에 & 연산자 없이 배열명을 포인터 배열의 초깃값으로 준다.



- 각 원소의 내용은 주소이므로 * 연산자를 기술했다면 각 1차원 배열의 첫 번째 원소 내용이 출력된다.

$p[0] == a == \&a[0]$ 이므로 $*p[0] == *a == *(\&a[0]) == a[0]$ 이 되고

$p[1] == b == \&b[0]$ 이므로 $*p[1] == *b == *(\&b[0]) == b[0]$ 이 되고

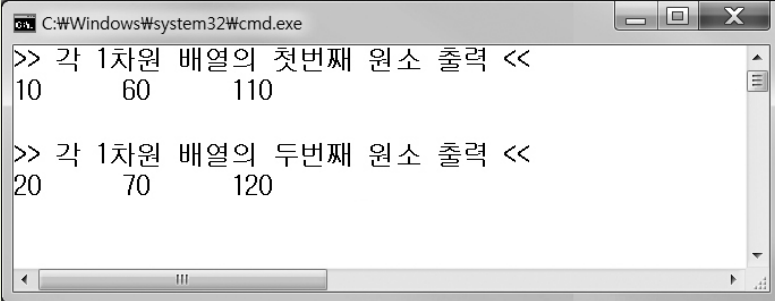
$p[2] == c == \&c[0]$ 이므로 $*p[2] == *c == *(\&c[0]) == c[0]$ 이 된다.

- $*p[0]$, $*p[1]$, $*p[2]$ 와 같은 포인터 표현은 $p[0][0]$, $p[1][0]$, $p[2][0]$ 와 같이 배열처럼 표현할 수 있으므로, 만일 각 1차원 배열의 두 번째 원소를 출력하면 다음과 같이 표현할 수 있다.

$p[0][1]$, $p[1][1]$, $p[2][1]$

예제 6-15. 포인터 배열에 1차원 배열의 주소값 저장하기(06_15.cpp)

```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05     int a[5]={ 10, 20, 30, 40, 50};
06     int b[5]={ 60, 70, 80, 90, 100};
07     int c[5]={110, 120, 130, 140, 150};
08
09     int *p[3]={a, b, c};
10
11     cout<<">> 각 1차원 배열의 첫번째 원소 출력 << \n";
12     cout<<p[0][0]<<"\t"<<p[1][0]<<"\t"<<p[2][0]<<"\n\n";
13
14     cout<<">> 각 1차원 배열의 두번째 원소 출력 << \n";
15     cout<<p[0][1]<<"\t"<<p[1][1]<<"\t"<<p[2][1]<<"\n";
16 }
```



```
C:\Windows\system32\cmd.exe
>> 각 1차원 배열의 첫번째 원소 출력 <<
10      60      110

>> 각 1차원 배열의 두번째 원소 출력 <<
20      70      120
```

03 2차원 포인터

③ 2차원 배열과 포인터 변수

- 2차원 배열 원소의 주소값을 출력하려면?
 - 배열 원소의 주소값을 출력하려면 배열에 첨자를 지정한 원소에 &을 붙이면 된다.
즉, `a[0][0]`의 주소값 == `&a[0][0]`

■ 2차원 배열에서의 배열명

- * 연산자를 두 번 붙이면 배열의 첫 번째 원소가 출력된다.
`**a==a[0][0];`
- ex) 2차원 배열 "int a[3][4]" 배열명에 포인터 연산자 +를 사용해 보자.
 - `a+1`은 2차원 배열의 시작 주소보다 16 (4x4) 바이트 큰 주소가 구해지고, `a+2`는 32 (4x4x2) 바이트 큰 주소가 구해진다. 이렇게 16바이트씩 증가한다는 것은 행 단위로 주소를 계산한다는 의미가 된다.
배열명이 2차원 포인터이므로 더하기 연산을 한 결과 역시 2차원 포인터가 되며, 이 포인터는 행 단위의 주소를 계산할 수 있는 2차원 포인터다.

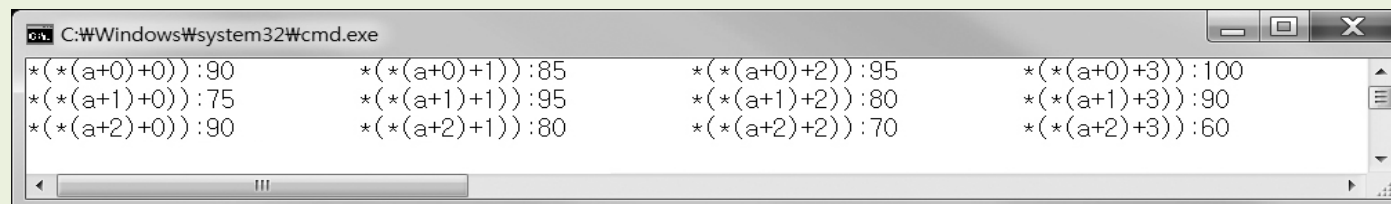
03 2차원 포인터

■ 2차원 배열의 포인터 연산

- 2차원 배열 "int a[r][c]"의 배열명에 포인터 관련 연산자인 *, + 만을 사용해 배열의 원소를 얻어낼 수 있다.
- $a[r][c] == *(*(a+r) + c)$ // 포인터 값의 행 단위 덧셈 후 열 단위 덧셈
- a가 2차원 배열명이므로 a는 2차원 포인터다. $*(a+r)+c$ 는 r번째 행의 시작 주소를 기준으로 4바이트(int)씩 c번 떨어진 위치의 주소를 계산한다. 계산된 주소에 * 연산자를 한번 더 붙여 $(*(a+r)+c)$ 그 위치의 값을 얻는다.

예제 6-19. 배열의 원소를 포인터 연산자를 이용해서 출력하기(06_19.cpp)

```
01 #include <iostream>
02 using namespace std;
03 #define ROW 3
04 #define COL 4
05 void main()
06 {
07     int a[ROW][COL] = { {90, 85, 95, 100},
08                          {75, 95, 80, 90},
09                          {90, 80, 70, 60}
10                      };
11     int r, c;
12     for(r=0; r<ROW; r++){
13         for(c=0; c<COL; c++) {
14             cout<<"*(" <a+<r<<" <+<" <c<<" <+<")<:<" <*<(" <a+r<+<" <c<<" <+<" <Wt<";
15         }
16         cout<<"\n";
17     }
18 }
```



```
C:\Windows\system32\cmd.exe
*(*(<a+0)+0)):90      *(*(<a+0)+1)):85      *(*(<a+0)+2)):95      *(*(<a+0)+3)):100
*(*(<a+1)+0)):75      *(*(<a+1)+1)):95      *(*(<a+1)+2)):80      *(*(<a+1)+3)):90
*(*(<a+2)+0)):90      *(*(<a+2)+1)):80      *(*(<a+2)+2)):70      *(*(<a+2)+3)):60
```

Homework

- Chapter 6 Exercise: 15, 17, 18, 19, 20, 22, 24, 26