# C++ / JAVA 프로그래밍

07주차 배열과 포인터

# 1차원 배열의 이해

- 1차원 배열
  - 배열에는 데이터를 여러 개 저장할 수 있으며, 배열에 저장된 각각의 값을 배열의 원소(element)라고 한다.

배열 선언 방법

**자료형 배열이름[원소의 개수];**

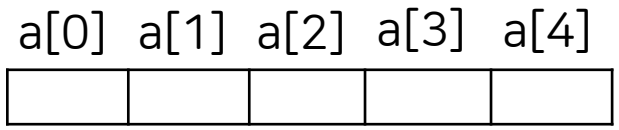
각 원소에 저장할 값에 대한 자료형

int a [ 5 ] ;

a: 배열명

5: 배열 원소의 개수

5개의 원소를 갖는 배열 선언
각각의 원소는 정수형
배열의 이름은 a




배열은 항상 0번부터 시작
n개의 원소가 있는 배열은 0번부터 n-1번까지 숫자가 붙음

각 원소를 사용하려면, '배열이름[사용할 원소 번호]'를 변수이름 대신 사용

# 예제 1. 배열 선언 및 사용법

```
01  #include <iostream>
02  using namespace std;
03
04  int main() {
05      int a[5];
06      int i;
07      a[0]=50;
08      a[1]=100;
09      a[2]=150;
10      a[3]=200;
11      a[4]=250;
12      for(i=0; i<5; i++)
13          cout << a[i] << "\t " ;
14      cout << "\n";
15      return 0;
16  }
17
18
19
20
21
22
23
24
```

출력결과:

50 100 150 200 250

# 1차원 배열 초기화

- `int a[5];`는 정수를 5개 저장할 수 있는 배열을 선언하는 것

- 배열 선언 시 초깃값을 지정할 수도 있음
  - 1차원 배열에 서는 초깃값을 하나 이상 주기 때문에 각 원소의 값을 콤마로 구분해 나열
  - 집합형태로 표현하기 위해 중괄호({ })로 묶음

- 기본적인 형태
  - `int a[5] = {1, 2, 3, 4, 5};`

# 1차원 배열 초기화 예시

- 배열에 초깃값이 지정되어 있을 경우에는 원소의 개수를 생략할 수 있음
  - 이 경우에는 컴파일러가 초깃값의 개수로 원소의 개수를 결정함
  - `int a[] = {10, 20, 30}` // 초깃값 개수가 3개이므로, 원소의 개수는 3이 됨
- 배열의 개수보다 초깃값이 적을 경우에는 나머지 영역에 0을 채움
  - `int a[5] = {10, 20, 30}` // 초깃값이 결정되지 않은 2개 원소는 0으로 초기화 됨
- 배열의 개수보다 초깃값을 많이 주었을 경우에는 에러가 발생함
  - `int a[5] = {10, 20, 30, 40, 50, 60, 70}` //에러
- 배열의 초깃값을 주지 않은 상태에서는 배열의 원소 개수를 생략할 수 없음
  - 배열이 초기화되지 않았다면 원소의 개수를 반드시 결정해 주어야 함
  - `int a[];` //에러

# 예제 2. 배열을 이용해 총합과 평균 구하기

```
01  #include <iostream>
02  using namespace std;
03
04  int main() {
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      return 0;
18  }
19
20
21
22
23
24
```

출력결과:

총합 = 445
평균 = 89

# 예제 3. 배열을 이용해 최댓값 구하기

```
01  #include <iostream>
02  using namespace std;
03
04  int main() {
05      int a[5];
06      int max;
07      int i;
08      cout << "정수형 데이터를 5개 입력하라\n";
09
10      for(i=0; i<5; i++){
11          cout << "a[ " << i <<" ] -> ";
12          cin >> a[i];
13      }
14
15      max = a[0];
16
17      for(i=1; i<5; i++)
18          if(a[i] > max)
19              max=a[i];
20
21      cout << "최댓값 : " << max <<"\n";
22      return 0;
23  }
24
```

출력결과:

정수형 데이터를 5개 입력하라
a[ 0 ] -> 10
a[ 1 ] -> 100
a[ 2 ] -> 30
a[ 3 ] -> 20
a[ 4 ] -> 50
최댓값 : 100

# 2차원 배열의 이해

- 2차원 배열
  - 2차원 배열용 자료형이 따로 있는 것이 아니라 1차원 배열을 선언하는 형식에 행과 열의 개수만 추가로 지정함

**2차원 배열 선언 방법**

**자료형 배열이름[행의 개수][열의 개수];**

int a[3][4]; 3개 행마다 각각 4개의 원소를 갖는 배열 선언
각각의 원소는 정수형
배열의 이름은 a

| | 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;

행과 열은 모두 0번부터 시작함
1행 2열은 두 번째 행 세 번째 열에 있는 원소를 의미함
변수를 사용하듯이 값을 대입할 수 있음

# 2차원 배열 초기화

- 중괄호를 이용해 집합을 원소로 갖는 집합을 표현
  - 바깥쪽 중괄호는 초기화를 위한 값을 집합 하나로 묶기 위한 것
  - 안쪽의 중괄호는 몇 번째 행에 대한 초기화인지를 구분하기 위한 것

```
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
              };
```

- 이중 for문을 이용한 2차원 배열의 출력

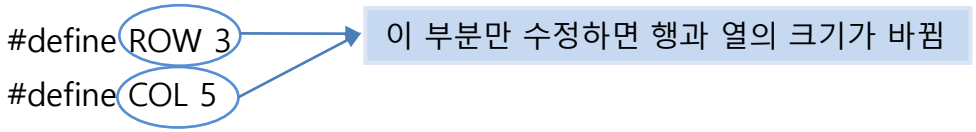
```
cout <<"\t" << a[0][0];
cout <<"\t" << a[0][1];
cout <<"\t" << a[0][2];
cout <<"\t" << a[0][3];
cout <<"\n";
cout <<"\t" << a[1][0];
cout <<"\t" << a[1][1];
          :
          :
cout <<"\t" << a[2][2];
cout <<"\t" << a[2][3];
```

=

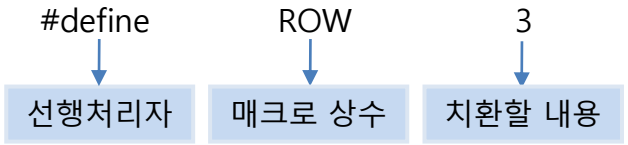
```
for(r=0; r<3; r++) {
    for(c=0; c<4; c++)
        cout<<"\t"<<a[r][c];
        cout<<"\n";
}
```

# #define문을 이용한 매크로 상수 정의

- 매크로 상수를 이용해서 행과 열의 개수를 정의하면 프로그램을 쉽게 수정할 수 있음




- 매크로 상수를 정의하는 방법




- 전처리자가 프로그램에서 사용한 매크로 상수를 만나면, 매크로를 정의할 때 지정한 문자열로 대치함

| 전처리 이전 | 전처리 이후 |
|---|---|
| #define ROW 3<br>#define COL 4<br>int main() {<br>  int a[**ROW**][**COL**];<br>} | #define ROW 3<br>#define COL 4<br>int main() {<br>  int a[**3**][**4**];<br>} |

# 예제 4. 2차원 배열 초기화 및 출력

```
01  #include <iostream>
02  using namespace std;
03  #define ROW 3
04  #define COL 4
05
06  int main() {
07      int a[ROW][COL] = {
08          {90, 85, 95, 100},
09          {75, 95, 80, 90},
10          {90, 80, 70, 60}
11      };
12      int r, c;
13      cout<<"\n 이중 for문으로 배열을 출력";
14      cout<<"\n ------------------------------\n";
15
16      for(r= 0; r<ROW; r++) {
17          for(c= 0; c<COL; c++)
18              cout<<"\t"<<a[r][c];
19          cout<<"\n";
20      }
21      return 0;
22  }
23
24
```

출력결과:

```
 이중 for문으로 배열을 출력
 ------------------------------
        90      85      95      100
        75      95      80      90
        90      80      70      60
```

# 예제 5. 두 행렬의 합 구하기

```
01  #include <iostream>
02  using namespace std;
03
04  int main() {
05      int a[3][4]={ {10, 20, 30, 40}, {20, 40 ,60 ,80}, {10, 30, 50, 70} };
06      int b[3][4]={ { 1, 2, 3, 4}, { 5, 6, 7, 8}, { 9, 10 ,11 ,12} };
07      int c[3][4];
08      int row, col;
09
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      cout<<" 두 행렬의 합을 출력하기";
14      cout<<"\n==================================\n";
15      for(row = 0; row < 3; row++){
16          for(col = 0; col < 4 ; col++)
17              cout<<" "<<c[row][col];
18          cout<<'\n';
19      }
20      return 0;
21  }
22
23
24
```

출력결과:

```
 두 행렬의 합을 출력하기
==================================
 11 22 33 44
 25 46 67 88
 19 40 61 82
```

# 2차원 배열 간단하게 초기화 하기

- 2차원 배열을 선언하면 행과 열의 수를 곱한 개수만큼의 기억공간이 생김

- 기억공간 전체를 일일이 지정하지 않고 간단하게 0으로 초기화하는 방법은 다음과 같음
  - int a[ROW][COL] = {0, };

- 기억 공간 전체에 값을 지정하지 않고 원소 하나에만 값을 주면 나머지 기억공간이 모두 0으로 채워짐

# 1차원 배열과 포인터

- 배열의 주소값을 출력하려면?
  - 배열 원소의 주소값은 배열에 첨자를 지정한 원소에 &을 붙이면 됨

- 배열명과 포인터
  - 배열명만 기술하면 C++ 컴파일러는 배열의 시작 주소값, 즉 포인터로 해석함
  - a == &a[0]

# 예제 6. 배열의 주소 알아보기

```
01  #include <iostream>
02  using namespace std;
03
04  int main() {
05      int a[5] = {10, 20, 30, 40, 50};
06      int i;
07
08      cout << "원소의 값을 출력 \n";
09      for(i=0; i<5; i++)
10          cout << a[i] << " " ;
11      cout << "\n";
12
13      cout << "\n원소의 주소 값을 출력 \n";
14      for(i=0; i<5; i++)
15          cout << &a[i] << "\n" ;
16
17      cout << "\n배열이름의 주소 출력 \n";
18      cout << " a :" << a << "\n";
19      cout << " &a[0] :" << &a[0] << "\n";
20
21      return 0;
22  }
23
24
```

출력결과:

```
원소의 값을 출력
10 20 30 40 50

원소의 주소 값을 출력
0x7ffffcb0a920
0x7ffffcb0a924
0x7ffffcb0a928
0x7ffffcb0a92c
0x7ffffcb0a930

배열이름의 주소 출력
 a :0x7ffffcb0a920
 &a[0] :0x7ffffcb0a920
```

# 배열의 이름과 포인터 연산

- 배열 이름은 배열의 시작 주소값을 알려주는 포인터
  - a == &a[0]
- *, & 연산자는 +, - 연산자와 같이 서로 반대되는 개념
  - 연속적으로 연산자를 기술하면 그 의미가 상쇄됨
  - *x와 같이 변수이름 앞에 * 연산자를 붙이면 그 주소를 찾아가 해당 기억공간에 저장된 값을 알려줌
  - 배열명에 * 연산자를 붙이면 배열의 시작 주소에 * 연산자를 붙인 것과 같으므로 배열의 첫 번째 원소값을 알려줌
  - *a == *&a[0] == a[0]
- 포인터 산술 연산

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

  - 배열의 첨자가 i인 원소의 주소는 a[i]앞에 & 연산자를 붙일 수도 있지만 a+i와 같이 배열명에 i를 더해 구할 수 있음
  - a+i == &a[i];

# 예제 7. 배열이름과 산술연산자

```
01  #include <iostream>
02  using namespace std;
03
04  int main() {
05      int a[5] = {10,20,30,40,50};
06      int i;
07      cout<<"배열 이름에 산술 연산자 사용하기\n";
08      for(i=0; i<5; i++){
09          cout<<" a+" << i << ": " << a+i;
10          cout<<" \t &a["<< i<<"]: "<<&a[i]<<"\n" ;
11      }
12      cout<<"\n포인터를 사용해서 배열의 원소값 출력하기\n";
13      for(i=0; i<5; i++){
14          cout<<" a[" << i << "]: "<<a[i];
15          cout<<" \t *(a+" << i << "): "<< *(a+i)<<"\n";
16      }
17      cout<<"\n포인터를 변수를 사용해서 배열의 원소값 출력하기\n";
18      int* p = a;
19      for(i=0; i<5; i++){
20          cout<<" *(p+" << i << "): "<<*(p+i);
21          cout<<" \t a[" << i << "]: "<< a[i]<<"\n";
22      }
23      return 0;
24  }
```

출력결과:

```
배열 이름에 산술 연산자 사용하기
 a+0: 0x7fffe27a8d30     &a[0]: 0x7fffe27a8d30
 a+1: 0x7fffe27a8d34     &a[1]: 0x7fffe27a8d34
 a+2: 0x7fffe27a8d38     &a[2]: 0x7fffe27a8d38
 a+3: 0x7fffe27a8d3c     &a[3]: 0x7fffe27a8d3c
 a+4: 0x7fffe27a8d40     &a[4]: 0x7fffe27a8d40

포인터를 사용해서 배열의 원소값 출력하기
 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

포인터를 변수를 사용해서 배열의 원소값 출력하기
 *(p+0): 10      a[0]: 10
 *(p+1): 20      a[1]: 20
 *(p+2): 30      a[2]: 30
 *(p+3): 40      a[3]: 40
 *(p+4): 50      a[4]: 50
```

# 2차원 포인터

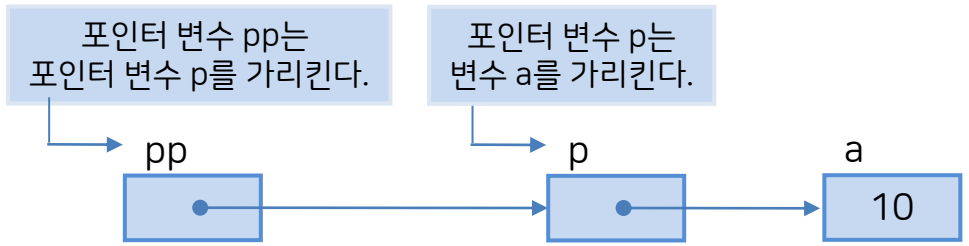
- 포인터의 포인터
  - 1차원 포인터의 주소값을 저장하는 포인터는 * 기호를 두 번씩 기술해서 선언한다.

2차원 포인터 선언 방법

**자료형** 포인터_변수_이름**

  - ex) int** pp;
  - 포인터와 관련된 연산자인 &과 *는 +와 – 사이 같은 관계
  - & 연산자를 변수 앞에 붙이면 차원이 점점 올라감
  - 반대로 * 연산자를 변수 앞에 붙이면 차원이 점점 내려감
  - 주의할 점: * 연산자는 중복해서 붙일 수 있지만, & 연산자는 변수 앞에 한 번만 붙일 수 있음

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



# 예제 8. 배열이름과 산술연산자

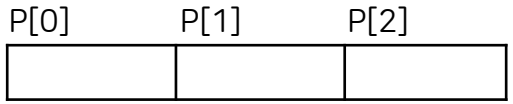
```
01  #include <iostream>
02  using namespace std;
03
04  int main() {
05      int a=5;
06      int *p;
07      int **pp;
08
09      p=&a;
10      pp=&p;
11
12      cout<<" p : "<< p <<" \t &a : "<< &a << endl;
13      cout<<" *p : "<< *p <<" \t \t a : "<< a << endl;
14      cout<<" pp : "<< pp <<" \t &p : "<< &p << endl;
15      cout<<" *pp : "<< *pp <<" \t p : "<< p << endl;
16      cout<<" **pp: "<< **pp <<" \t \t *p : "<< *p << endl;
17      return 0;
18  }
19
20
21
22
23
24
```

출력결과:

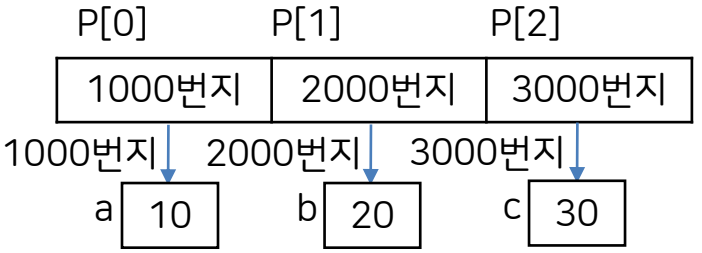
```
p : 0x7fffc74afb64       &a : 0x7fffc74afb64
*p : 5                   a : 5
pp : 0x7fffc74afb68      &p : 0x7fffc74afb68
*pp : 0x7fffc74afb64     p : 0x7fffc74afb64
**pp: 5                  *p : 5
```

# 포인터 배열

- 1차원 포인터를 저장하는 포인터 배열
  - int *p1, *p2, *p3;  // 3개의 정수형 포인터 변수 선언
  - int *p[3];  // 3개의 정수형 포인터를 저장할 수 있는 배열 선언, 각 원소에는 1차원 포인터 저장




- 각 원소에 1차원 포인터를 저장한다면
  - 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 함
  - p[0]==&a 이므로 *p[0]==a

# 예제 9. 배열이름과 산술연산자

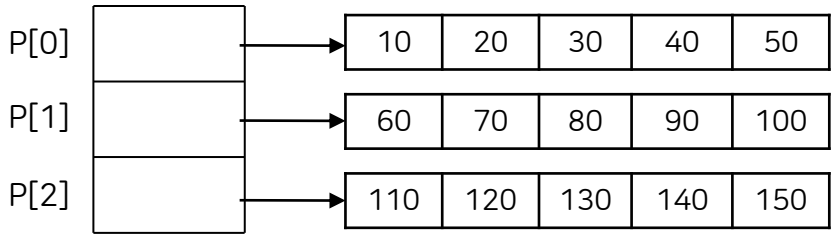
```
01  #include <iostream>
02  using namespace std;
03
04  int main() {
05      int a=10, b= 20, c=30; // 정수형 변수
06
07      // 포인터 배열에 변수의 주소를 저장해 둔다.
08      int *p[3]={&a, &b, &c};
09
10      // 배열 원소에 * 연산자로 정수값을 얻어온다.
11      cout<<"\n *p[0] : "<< *p[0];
12      cout<<"\t *p[1] : "<< *p[1];
13      cout<<"\t *p[2] : "<< *p[2];
14
15      // * 연산자 대신 []로 정수값을 얻어온다.
16      cout<<"\n p[0][0] : "<< p[0][0];
17      cout<<"\t p[1][0] : "<< p[1][0];
18      cout<<"\t p[2][0] : "<< p[2][0];
19      cout<<"\n";
20      return 0;
21  }
22
23
24
```

출력결과:

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

# 포인터 배열로 2차원 배열 표현

- 배열명 자체가 포인터이기 때문에 & 연산자 없이 배열명을 포인터 배열의 초깃값으로 사용할 수 있음
  - 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};




- 각 원소의 내용은 주소이므로 * 연산자를 기술하면 각 1차원 배열의 첫 번째 원소 내용이 출력된다.
  - p[0]==a==&a[0]이므로 *p[0]==*a==*&a[0]==a[0]
  - p[1]==b==&b[0]이므로 *p[1]==*b==*&b[0]==b[0]
  - p[2]==c==&c[0]이므로 *p[2]==*c==*&c[0]==c[0]
- *p[0], *p[1], *p[2]와 같은 포인터 표현은 p[0][0], p[1][0], p[2][0]와 같이 사용할 수 있음
  - 만일 각 1차원 배열의 두번째 원소를 출력하려면 p[0][1], p[1][1], p[2][1]를 사용함

# 예제 10. 포인터 배열에 1차원 배열의 주소값 저장하기

```
01  #include <iostream>
02  using namespace std;
03
04  int main() {
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      int *p[3]={a, b, c};
09
10      cout<<">> 각 1차원 배열의 첫번째 원소 출력 << \n";
11      cout<<p[0][0]<<"\t"<<p[1][0]<<"\t"<<p[2][0]<<"\n\n";
12
13      cout<<">> 각 1차원 배열의 두번째 원소 출력 << \n";
14      cout<<p[0][1]<<"\t"<<p[1][1]<<"\t"<<p[2][1]<<"\n";
15      return 0;
16  }
17
18
19
20
21
22
23
24
```

출력결과:

```
>> 각 1차원 배열의 첫번째 원소 출력 <<
10      60      110

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

# 2차원 배열과 포인터 변수

- 배열 원소의 주소값을 출력하려면?
  - 배열에 첨자를 지정한 원소에 &을 붙이면 됨
  - a[0][0]의 주소값을 알고 싶으면 &a[0][0]이라고 하면 됨

- 2차원 배열에서 첨자를 생략하는 형태
  - 2차원 배열에서 첨자를 생략하는 형태는 2가지, 열만 생략하든지, 행과 열을 동시에 생략해야 함
  - 2차원 배열 int a[3][4];에 대해서 열만 생략하면 행이 3개가 된다.
    - a[0], a[1], a[2]
  - 2차원 배열에서 열을 생략하고 행만 지원하면 원소값 대신 주소가 출력되며, 이 주소들은 16 바이트씩 차이남
  - 행 1개가 열 4개로 구성되어 있고 정수형 배열이므로 16바이트 (4바이트×4개) 차이가 나는 것
  - 2차원 배열에서 열을 생략하면 각 행의 시작 주소를 의미하며, 아래 식의 n은 행의 위치를 알려주는 첨자임
    - a[n] == &a[n][0]

# 예제 11. 2차원 배열의 주소값 출력하기

```
01  #include <iostream>
02  using namespace std;
03  #define ROW 3
04  #define COL 4
05
06  int main() {
07      int a[ROW][COL] = {
08          {90, 85, 95, 100},
09          {75, 95, 80, 90},
10          {90, 80, 70, 60}
11      };
12      cout<<"2차원 배열에 저장된 원소들의 주소\n";
13      cout<<"-------------------------------------------------------";
14
15      for(int r= 0; r<ROW; r++){
16          cout<<"\n"<< r << "행 ";
17          for(int c= 0; c<COL; c++) {
18              cout<<"\t "<<&a[r][c]; // 배열의 주소값 출력
19          }
20      }
21      cout<<"\n";
22      return 0;
23  }
24
```

출력결과:

```
2차원 배열에 저장된 원소들의 주소
-------------------------------------------------------
0행      0x7fffc68f0de0  0x7fffc68f0de4  0x7fffc68f0de8  0x7fffc68f0dec
1행      0x7fffc68f0df0  0x7fffc68f0df4  0x7fffc68f0df8  0x7fffc68f0dfc
2행      0x7fffc68f0e00  0x7fffc68f0e04  0x7fffc68f0e08  0x7fffc68f0e0c
```

# 예제 12. 2차원 배열에 행만 지정해서 출력하기

```
01  #include <iostream>
02  using namespace std;
03  #define ROW 3
04  #define COL 4
05
06  int main() {
07      int a[ROW][COL] = {
08          {90, 85, 95, 100},
09          {75, 95, 80, 90},
10          {90, 80, 70, 60}
11      };
12
13      cout<<"2차원 배열의 각 행의 첫번째 열의 주소\n";
14      cout<<"-------------------------------------------------------";
15      for(int r= 0; r<ROW; r++){
16          cout<<" \n"<< r << "행 ";
17          cout<< "\t a[" << r << "] = " << a[r];
18          cout<< "\t &a[" << r << "][0] = " << &a[r][0];
19      }
20      cout<<"\n\n각 행의 첫번째 열에 위치한 원소\n";
21      cout<<"-----------------------------------------------\n";
22      cout<<" *a[0]="<<*a[0]<<"\t*a[1]="<<*a[1]<<"\t *a[2]="<<*a[2]<<"\n";
23      return 0;
24  }
```

출력결과:

```
2차원 배열의 각 행의 첫번째 열의 주소
-------------------------------------------------------
0행      a[0] = 0x7fffdafee060   &a[0][0] = 0x7fffdafee060
1행      a[1] = 0x7fffdafee070   &a[1][0] = 0x7fffdafee070
2행      a[2] = 0x7fffdafee080   &a[2][0] = 0x7fffdafee080

각 행의 첫번째 열에 위치한 원소
-----------------------------------------------
 *a[0]=90       *a[1]=75         *a[2]=90
```

# 2차원 배열에서의 배열명

- * 연산자를 두 번 붙이면 배열의 첫 번째 원소가 출력됨, 예: **a==a[0][0];
- 2차원 배열명에 포인터 연산자 + 사용하기
  - a+1이 2차원 배열의 시작주소보다 16바이트 큰 주소라면, a+2는 시작주소보다 32바이트 큰 주소가 됨
  - 16바이트씩 증가한다는 것은 행 단위로 주소를 계산한다는 의미
  - 배열명이 2차원 포인터이므로 더하기 연산을 한 결과 역시 2차원 포인터
- 2차원 배열의 원소에 도입한 포인터 개념
  - 2차원 배열명에 포인터 관련 연산자인 *, + 만을 사용해 배열의 원소를 사용해서 얻어낼 수 있음
    - a[ r ][ c ]==* ( * ( a + r ) +c )
  - a가 2차원 배열명이라면 여기에 r을 더해 a+r한 결과는 r번째 행의 시작주소를 의미하는 2차원 포인터
  - 여기에 계속 더하기를 하면 주소가 행 단위로 계산되므로 차원을 낮추기 위해 * 연산자를 붙임
  - *(a+r)+c는 r번째 행의 시작 주소를 기준으로 4바이트씩 c번 떨어진 위치의 주소를 계산
  - 최종적으로 계산된 주소인 *(a+r)+c에 * 연산자를 한 번 더 붙이면 그 위치의 값을 출력

# 예제 13. 2차원 배열명의 의미

```
01  #include <iostream>
02  using namespace std;
03  #define ROW 3
04  #define COL 4
05
06  int main() {
07      int a[ROW][COL] = { {90, 85, 95, 100},
08          {75, 95, 80, 90},
09          {90, 80, 70, 60}
10      };
11      cout<< " a : " << a <<"\n";
12      cout<< " *a : " << *a <<"\n";
13      cout<< " **a : " << **a <<"\n";
14      cout<<"-------------------------------------\n";
15      cout<< " a + 1 : " << a + 1 <<"\n";
16      cout<< " a + 2 : " << a + 2 <<"\n";
17      return 0;
18  }
19
20
21
22
23
24
```

출력결과:

```
 a : 0x7fffd7db6110
 *a : 0x7fffd7db6110
 **a : 90
-------------------------------------
 a + 1 : 0x7fffd7db6120
 a + 2 : 0x7fffd7db6130
```

# 예제 14. 포인터 연산자를 이용해 배열의 원소 출력하기

```
01  #include <iostream>
02  using namespace std;
03  #define ROW 3
04  #define COL 4
05
06  int main() {
07
08      int a[ROW][COL] = { {90, 85, 95, 100},
09          {75, 95, 80, 90},
10          {90, 80, 70, 60}
11      };
12      int r, c;
13      for(r=0; r<ROW; r++){
14          for(c=0; c<COL; c++)
15              cout<<"*(*(a+"<<r<< ")+"<<c<<")):"<<*(*(a+r)+c)<<" \t";
16      cout<<"\n";
17      }
18  }
19
20
21
22
23
24
```

출력결과:

```
*(*(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
```

# 2차원 포인터 변수는 어떻게 선언해야 하는가? (1/2)

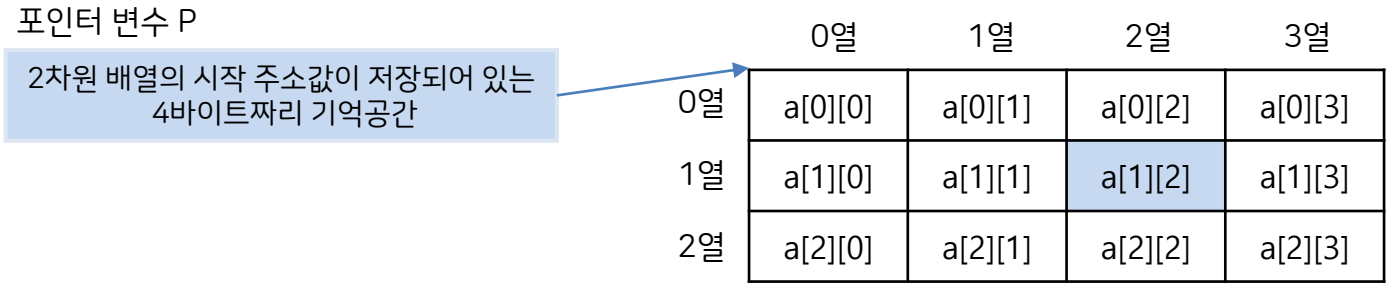
- 2차원 배열명을 int **p로 선언한 포인터 변수에 저장하면 에러가 발생한다.

```
int a[3][4];
int **p;
p=a;
---------------------------------------
Cannot convert 'int[4] * 'to 'int **'
```

- 2차원 포인터지만 2차원 배열을 가리키려면 각 행이 열 몇 개로 구성되었는지에 대한 정보가 있어야 함
  - int (*p)[4];
  - P=a;
- 열이 4개로 구성된 형태의 배열을 가리키는 포인터 변수 p를 선언하고, 여기에 2차원 배열명을 대입
  - 2차원 배열은 1차원 배열의 모임이라는 것을 기억하자!
- 2차원 배열의 시작 주소값인 a를 p에 대입했으므로 2차원 배열 a의 원소들은 다음과 같이 출력할 수 있음
  - *(*(p+r)+c)

# 2차원 포인터 변수는 어떻게 선언해야 하는가? (2/2)

- p는 선언 시 한 행이 열 4개로 구성된 배열을 가리키는 포인터로 선언했음
  - 따라서 p+1을 하면 16바이트(4*4)가 증가함
  - p+r한 결과는 r번째 행의 시작 주소값이고 2차원 포인터
  - 여기에 계속 더하기를 하면 행 단위로 주소가 계산되므로 차원을 낮추기 위해 * 연산을 붙인 후 더함
  - *(p+r)+c는 r번째 행의 시작 주소를 기준으로 4바이트씩 c번 떨어진 위치의 주소를 계산
  - 최종적으로 계산된 주소에 * 연산을 한 번 더 붙인*(*(p+r)+c)는 그 위치의 값을 출력




- 2차원 배열의 주소를 저장할 포인터 변수의 선언은 반드시 한 행에 포함된 열의 개수를 명시해야 함!


HANYANG
UNIVERSITY