7. 포인터(pointer)

포인터는 **메모리 주소**(번지, address)를 직접 다루는 변수이다.
기억공간의 주소값을 갖는 변수를 포인터 또는 포인터 변수라고 한다.
포인터는 연산자 *를 사용하여 선언한다.

```
char  *  ptr;
  |   |   |
  |   |   └──→ 포인터(포인터 변수)
  |   └──────→ ptr이 포인터임을 선언(간접지정연산자)
  └──────────→ 포인터형을 char형으로 선언
```

ptr : 메모리의 **주소값**을 가진다.(char형 포인터 값)
*ptr : 포인터 ptr이 가리키는 **주소에 수록된 자료**. 즉, 주소 ptr의 내용이다.
(포인터 앞에 *를 붙이면 내용물이 된다)

◈ 포인터 종류(포인터 형)

```
char  *p;   → char형 포인터 p
int   *q;   → int형 포인터 q
long  *r;   → long형 포인터 r
float *f; 등
```

---

① 포인터 p, q, r, f의 공통점은 메모리 주소값을 기억하고 **주소를 직접 다룰 수 있다.**
② 가장 큰 차이점은 포인터 형이 갖는 **바이트 수만큼씩 자료를 취급**하는 것이다.

◈ 포인터를 1 증가시켰을 때(p++) 메모리 실제 주소 증가분

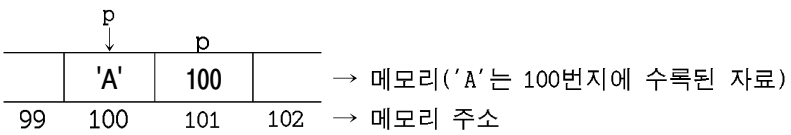
선 언	16비트 시스템	32비트 시스템
char *p;	1 바이트씩 증가	1 바이트씩 증가
int *p;	2 바이트씩 증가	4 바이트씩 증가
long *p;	4 바이트씩 증가	8 바이트씩 증가
float *p;	4 바이트씩 증가	4 바이트씩 증가

- **자료형 크기**는 OS(16비트, 32비트, 64비트)나 **컴파일러에 따라 조금씩 다르다.**
- 특히, long은 4byte로 취급하는 OS가 많다.**(OS windows : int와 long 둘 다 4byte로 취급)**
- long long : 8byte로 취급

◉ 포인터에 대한 기본 개념을 그림으로 설명

```
char  *p;
p = (char *)100;
*p = 'A';
```

일 때, 메모리 모습은 다음과 같다(컴파일은 불가능)



- 'p = (char *)100;'에서 '(char *)'는 형변환연산자로 사용된 부분이다.
- 형변환연산자는 자료형을 강제적으로 바꾸어 주는 연산자이다.
- 여기서는 100이라는 수를 char형 포인터로 바꾸어 준다는 뜻이다.

◈ 포인터와 주소연산자(&)

일반 변수가 위치하는 메모리의 주소를 구하기 위해서는 **주소연산자 &를 사용**한다.

예를 들어,

```
a = 7;
p = &a;
b = *p;
```

라고 기술했을 때,

◉ 포인터와 주소연산자 사용 예

```
void main(){
  int  a, b;
  int  *p;          //반드시 포인터로 선언해야 한다.

  a = 7;
  p = &a;           //p에는 변수 a의 주소가 대입
  b = *p;           //포인터 앞에 *가 있으면 내용물이다.
  printf("%d\n", b);
}
```

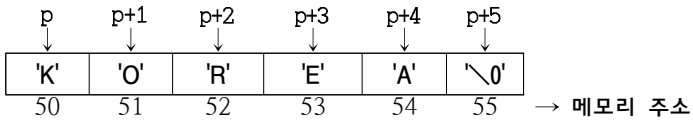
실행결과 : 7

◈ 포인터와 문자열

포인터를 이용하여 문자열을 다루는 방법을 살펴본다.(배열과 매우 비슷하다)

---

char *p = "KOREA"; 라고 정의하면 기억공간에 다음과 같은 모습이 된다.



---

① 앞에서 설명한 것처럼 C에서 문자열 끝에는 '\0'이 저장된다.
② 포인터 p는 배열명처럼 문자열 시작주소 값을 가진다.(매우 중요!)
③ 위의 그림을 기준으로 하면, p = 50이 되며 다음과 같은 의미가 된다.

```
*p = 'K'      ┐
*(p+1) = 'O'  │
*(p+2) = 'R'  │ 포인터 앞에 *가 있으면 포인터가 가리키는 주소의 내용이 된다.
*(p+3) = 'E'  │
*(p+4) = 'A'  ┘
```

◉ 포인터를 이용한 문자열 취급

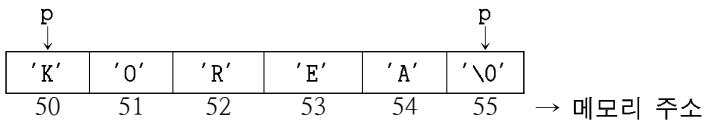
```
void main()
{
  char *p = "KOREA";
  printf( "%s\n", p );          //문자열 출력
  printf( "%s\n", p+2 );        //p+2 번지부터 문자열 출력
  printf( "%c\n", *p );         //p번지의 내용을 문자로 출력
  printf( "%c\n", *(p+2) );     //p+2번지의 내용을 문자로 출력
  printf( "%c\n", *p+2 );       //괄호가 없으면, *p + 2 = 'K' + 2 = 'M'
}
```

실행결과 : KOREA
REA
K
R
M

◈ 포인터를 이용한 주소 연산

포인터를 1 증가시키면, 포인터는 다음 기억장소를 가리킨다.

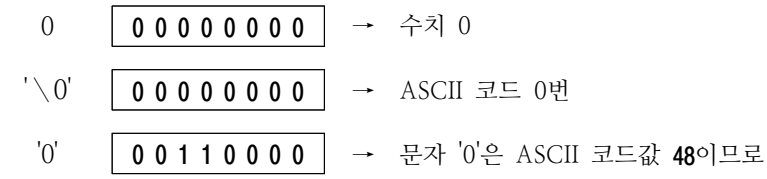
```
void main()
{
      char *p = "KOREA" ;
      for(  ; *p ;  ) putchar( *p++ );
}
//출력결과 : KOREA
```



- 위의 프로그램에서 for문의 조건식으로 '*p'가 사용되었다.
- 조건식 '*p'의 값이 0이 되면, 조건은 거짓이 되고, for문의 반복 실행이 종료하게 된다.
- 출력문의 '*p++'가 반복 실행되면서 포인터 p는 마지막에 있는 '\0'을 가리키게 된다.
- 마지막에 있는 '\0'은 수치 0과 같다. 조건은 거짓이 되고, for문의 반복은 종료한다.

C에서 0, '\0', '0'의 개념(차이점)

0, '\0', '0'으로 표현되는 값들의 차이점을 메모리 구조를 이용하여 이해하도록 한다.



- 수치 0과 문자 '0'은 엄연히 다르다.
- Escape sequence 방식으로 표현된 '\0'은 ASCII 코드 0번을 의미한다.
- '\0'은 수치 0과 같은 개념이다.(즉, 수 0과 '\0'은 같다)

◈ 포인터와 증감연산자 결합

*p++	• 먼저, p번지의 내용물을 구하고 • 포인터 p를 1증가시킨다. → ++가 포인터 p에 붙어 있으므로 • *p++는 *(p++)와 같다. • ( )가 있다고 p++가 먼저 수행되는 것은 아니다. 주의할 것!
*++p	• 먼저, 포인터 p를 1증가시키고 p번지의 내용물을 구한다. • *++p는 *(++p)와 같다.
(*p)++	• 먼저, p번지의 값(*p)을 먼저 구하고, 나중에 p번지의 값 자체를 1증가시킨다. • 포인터 p의 값은 증가되지 않는다.
++*p	• p번지의 값을 구하고 우선적으로 *p의 값 자체를 1증가시킨다. • 포인터 p의 값은 증가되지 않는다. • ++*p는 ++(*p)와 같다.

↓

↓ 예를 들면

↓

```
void main(){
    int *p;
    int a[] = {10, 20, 30, 40};
    p = a;
    printf("%d  ", *++p);
    printf("%d  ", *p++);
    printf("%d  ", *p);
}
```
실행 결과 : 20 20 30

```
void main(){
    int *p;
    int a[] = {10, 20, 30, 40};
    p = a;
    printf("%d  ", *++p);
    printf("%d  ", *++p);
    printf("%d  ", *p);
}
```
실행 결과 : 20 30 30

```
void main(){
    int *p;
    int a[] = {10, 20, 30, 40};
    p = a;
    printf("%d  ", *++p);
    printf("%d  ", (*p)++ );
    printf("%d  ", *p);
}
```
실행 결과 : 20 20 21

```
void main(){
    int *p;
    int a[] = {10, 20, 30, 40};
    p = a;
    printf("%d  ", *++p);
    printf("%d  ", ++*p);
    printf("%d  ", *p);
}
```
실행 결과 : 20 21 21

[Tip] 다음과 같은 문장을 이해하는 방법

*++p : 단항연산자 ++가 포인터 p에 붙어 있으므로 p가 증가된다.

++*p : 단항연산자 ++가 *p에 붙어 있으므로 p가 가리키는 번지의 내용이 증가된다.

◈ 배열과 포인터(같은 점과 차이점)

배열과 포인터는 메모리의 주소를 기억하게 된다.(같은 점)

```
char a[ ] = "KOREA";
     └──────►  배열명 a에는 배열의 시작주소가 기억된다.

char *p = "SEOUL";
     └──────► 포인터 p에는 문자열의 시작주소가 기억된다.
```

---

① 포인터와 배열명은 둘 다 주소값을 가지면서 서로 **부분적으로 호환성**도 있다.
② 그러나, 다른 점은
- **포인터** : 주소연산이 **가능**하다. (변수, p++; 가능)
- **배열명** : 주소연산이 **불가능**하다.(상수, a++; 불가)

③ 즉, 배열명은 항상 선언된 배열의 시작주소값만을 가진다.
배열명은 포인터처럼 기억된 주소값을 변경할 수 없다.
④ 배열명은 주소값을 갖는 **상수(포인터 상수)**의 개념이다.

◉ 배열과 포인터를 이용한 문자열 취급(호환성이 있다)

```
void main()
{
     char a[] = "KOREA";
     char *p = "SEOUL";

     printf("%c\n", *a);        //배열로 정의된 것을 포인터 방식으로 사용
     printf("%c\n", p[0]);      //포인터로 정의된 것을 배열 방식으로 사용
     printf("%s\n", a+2);
     printf("%s\n", &p[2]);     //p+2 번지부터 문자열 출력
}
```

실행결과 :
```
K
S
REA
OUL
```

◈ 배열의 배열

① C는 1차원 배열만 지원한다.

- 정확하게 이야기하면 C에는 다차원 배열이 존재하지 않는다.
- 단지, 통상적으로 과거 다른 언어에서 표현하는 방법으로 이야기할 뿐이다.

② 그 대신에 배열요소의 데이터 형에는 제한이 없다.

- 이것은 '배열요소 자체가 배열인 구조로 다차원 배열구조를 구현한다'라는 뜻이다.

다음은 C의 배열 배열구조를 보여주고 있다.

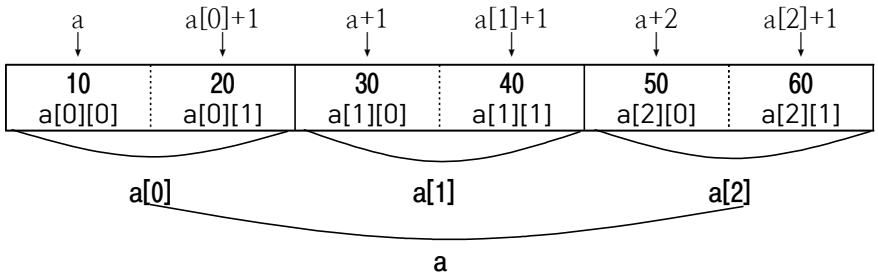
```
void main()
{
  int a[3][2] = {10,20,30,40,50,60};

  printf("%d %d %d %d \n",a[0][0],*a[0],     *(*a+0),**a );
  printf("%d %d %d %d \n",a[0][1],*(a[0]+1),*(*a+1),*((*a)+1) );
  printf("%d %d %d %d \n",a[1][0],*a[1],     *(*a+2),**(a+1) );
  printf("%d %d %d %d \n",a[1][1],*(a[1]+1),*(*a+3),*(*(a+1)+1) );
  printf("%d %d %d %d \n",a[2][0],*a[2],     *(*a+4),**(a+2) );
  printf("%d %d %d %d \n",a[2][1],*(a[2]+1),*(*a+5),*(*(a+2)+1) );
}
```

[실행결과]

```
10 10 10 10
20 20 20 20
30 30 30 30
40 40 40 40
50 50 50 50
60 60 60 60
```

- 배열 a는 배열요소 a[0], a[1], a[2]를 가지는 배열의 배열인 구조이다.



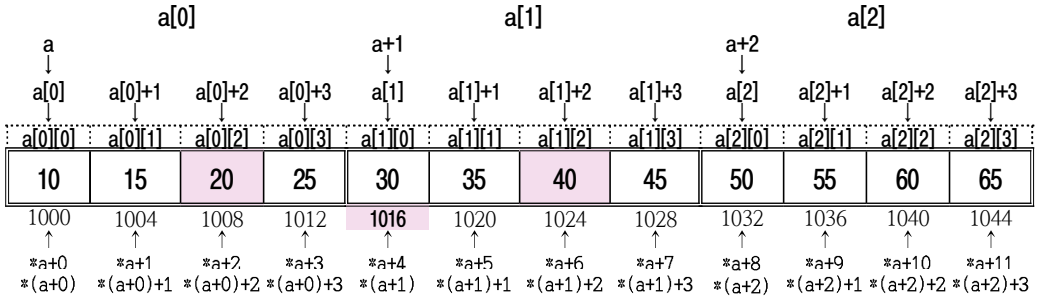
예제	다음 C 프로그램의 출력 값으로 옳은 것은? [2017년 국가 7급 유형] (단, int 데이터 타입은 4byte로 표현되며, 배열 a의 시작주소는 1,000이다)

```
-----------------------------------------------------------------
void main()
{
   int a[][4] = {10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65};
   printf("a+1 = %d, ", a+1);                    // 출력 : 1016
   printf("*(*a+2) = %d, ", *(*a+2));            // 출력 : 20
   printf("*(*(a+1)+2) = %d\n", *(*(a+1)+2));    // 출력 : 40
}
-----------------------------------------------------------------
```

↓

↓ 메모리 구조

↓



◇ a+1 = 1016 → int 데이터 타입의 크기는 4byte이므로

◇ *(*a+2) = 20

◇ *(*(a+1)+2) = 40

// 배열의 배열에서 내용물을 나타내는 경우

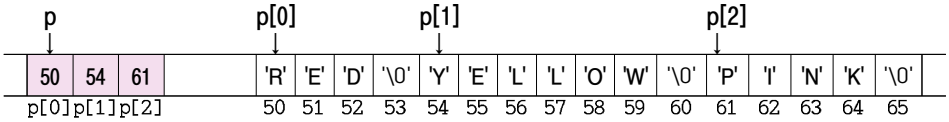
대괄호가 2개 있는 경우	a[1][1]
별이 2개 있는 경우	*(*(a + 1) + 1)
괄호와 별이 각각 1개 있는 경우	*(a[1] + 1)

◈ 포인터 배열

먼저, "포인터 배열과 배열의 배열"의 차이는 다음과 같다.

◈ 포인터 배열

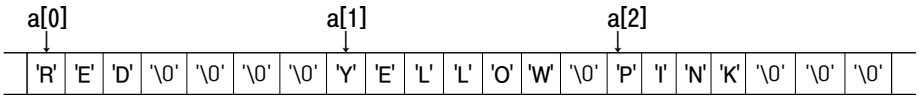
```
char *p[] = { "RED", "YELLOW", "PINK" };
```



- p는 배열의 시작주소 값을 가지고,
- p[0], p[1], p[2]는 각각 문자열이 저장된 시작주소 값을 갖는다.

◈ 배열의 배열(2차원 배열 개념)

```
char a[ ][7] = { "RED", "YELLOW", "PINK" };
```



- a와 a[0]은 같은 주소 값을 가진다.(배열명은 배열의 시작주소 값을 가지므로)
- 길이가 짧은 문자열 뒤에는 불필요한 '\0'이 많이 저장된다.(메모리 낭비)

---

① 포인터 배열은 배열요소가 메모리 주소를 기억하는 구조이다.
② 길이가 다른 문자열을 취급하는데 사용하면 메모리 낭비가 없어진다.
③ 포인터 배열은 배열과 포인터의 특징을 동시에 갖는다.
④ 포인터 배열 p와 배열의 배열 a는 둘 다 배열명이므로 주소연산이 불가능하다.

◉ 길이가 다른 여러 개의 문자열을 다루는 포인터 배열

```
void  main(){
      char *p[] = {"RED", "WHITE", "BLUE"};

      printf("%s\n", p[0]);                    //문자열 출력
      printf("%s\n", p[1]);
      printf("%s\n", p[2]);
}
```

실행결과 : RED
WHITE
BLUE

기출문제 분석

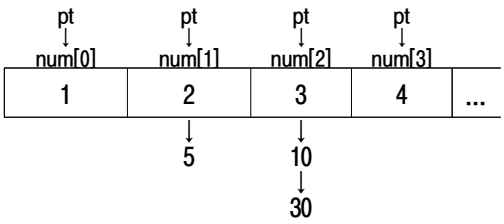
1. 다음의 C 프로그램을 실행한 결과로 옳은 것은? [2017년 서울 9급]

```
#include <stdio.h>
void main(){
  int num[4] = {1, 2, 3, 4};
  int *pt = num;
  pt++;
  *pt++ = 5;
  *pt++ = 10;
  pt--;
  *pt++ += 20;
  printf("%d %d %d %d", num[0], num[1], num[2], num[3]);
}
```

① 1 5 10 20　　② 1 5 20 4
③ 1 5 30 4　　④ 에러 발생

♤ 배열과 포인터

// 메모리 구조



`int *pt = num;` → pt가 num[0]을 가리킴
`pt++;` → pt가 num[1]을 가리킴
`*pt++ = 5;` → num[1] = 5, pt++, pt가 num[2]를 가리킴
`*pt++ = 10;` → num[2] = 10, pt++, pt가 num[3]을 가리킴
`pt--;` → pt--, pt가 num[2]를 가리킴
`*pt++ += 20;` → num[2] = num[2] + 20 = 10 + 20 = 30

정답 : ③

2. 다음 C 프로그램에서 밑줄 친 코드의 실행 결과와 동일한 결과를 출력하는 코드로 옳은 것만을 모두 고르면? [2022년 국가 9급]

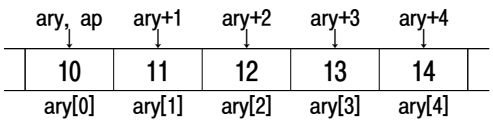
```
#include <stdio.h>
int main()
{
  int ary[5] = {10, 11, 12, 13, 14};
  int *ap;
  ap = ary;
  printf("%d", ary[1]);
  return 0;
}
```

ㄱ. printf("%d", ary+1);　　ㄴ. printf("%d", *ap+1);
ㄷ. printf("%d", *ary+1);　　ㄹ. printf("%d", *ap++);

① ㄱ, ㄴ　② ㄴ, ㄷ
③ ㄴ, ㄹ　④ ㄴ, ㄷ, ㄹ

♤ C 프로그램 - 배열과 포인터

```
int main()
{
  int ary[5] = {10, 11, 12, 13, 14};
  int *ap;
  ap = ary;
  printf("%d", ary[1]);  // 출력 : 11
  return 0;
}
```



ㄱ. printf("%d", ary+1);　// 출력 : 2번째 배열원소 주소 출력
ㄴ. printf("%d", *ap+1);　// 출력 : 10 + 1 = 11
ㄷ. printf("%d", *ary+1);　// 출력 : 10 + 1 = 11
ㄹ. printf("%d", *ap++);　// 출력 : 10

정답 : ②

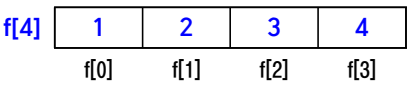
3. C 언어로 작성된 프로그램의 실행 결과로 옳은 것은? [2019년 우정 9급]

```
#include <stdio.h>
double h(double *f, int d, double x)
{
  int i;
  double res = 0.0;
  for(i=d-1; i>=0; i--)
    res = res * x + f[i];
  return res;
}
int main(){
  double f[] = {1, 2, 3, 4};
  printf("%3.1f\n", h(f, 4, 2));
  return 0;
}
```

① 11.0 ② 26.0
③ 49.0 ④ 112.0

♤ C 프로그램 - 배열과 포인터

// 배열구조



- res = res * x + f[i];
  ↓
  ↓ x = 2, i = 3, 2, 1, 0
  ↓
- res = res * x + f[3] = 0.0 * 2 + 4.0 = 4.0
- res = res * x + f[2] = 4.0 * 2 + 3.0 = 11.0
- res = res * x + f[1] = 11.0 * 2 + 2.0 = 24.0
- res = res * x + f[0] = 24.0 * 2 + 1.0 = **49.0**

정답 : ③

4. 다음 C 프로그램의 실행 결과는? [2017년 경기 추가 9급]

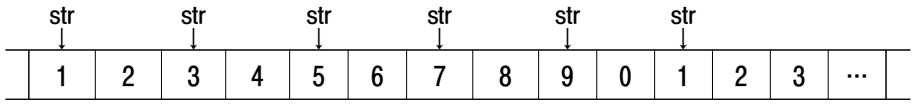
```
#include <stdio.h>
int main(void)
{
  int i;
  char buf[]="123456789012345678901234567890";
  char *str, ch;
  str = buf;
  for(i = 0; i <= 20; i += 4)
  {
    printf("%c ", *str++);
    ch = *++str;
  }
  return 0;
}
```

① 1 3 5 7 9 1　　② 1 5 9 3 7 1

③ 2 4 6 8 0 2　　④ 2 6 0 4 8 2

♤ 단항연산자와 포인터

- printf("%c ", *str++);
  → 포인터 str이 가리키는 주소 내용을 출력 후, 포인터 str 증가
- ch = *++str;
  → 포인터 str 증가 후, str이 가리키는 주소 내용을 ch에 대입




```
for(i = 0; i <= 20; i += 4) //6번 반복
{
  printf("%c ", *str++);      //1부터 시작해서 하나씩 건너가면서 출력(1 3 5 7 9 1)
  ch = *++str;               //그냥 포인터 str만 1 증가하는 역할만 할 뿐이다.
}
```

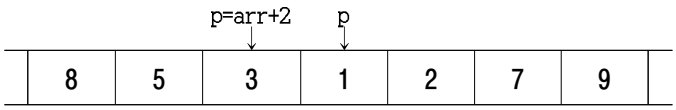
정답 : ①

5. 다음 프로그램의 실행 결과는? [2015년 국회 9급]

```
#include <stdio.h>
int main()
{
  int arr[] = {8, 5, 3, 1, 2, 7, 9};
  int *p = arr+2, a = 0, b = 0;
  a = *++p;
  b = (*p)++;
  printf("%d, %d\n", a, b);
  return 0;
}
```

① 3, 3　② 3, 1　③ 1, 1
④ 1, 2　⑤ 4, 1

♤ **배열과 포인터**



① *++p는 *(++p)와 같다.
- 먼저, 포인터 p를 1증가시키고
- p번지의 내용물을 구한다.

② (*p)++
- 먼저, p번지의 값(*p)을 구하고, 나중에 p번지의 값 자체를 1증가시킨다.
- 포인터 p의 값은 증가되지 않는다.

a = *++p; → 먼저, 포인터 p를 1증가시키므로 a = 1
b = (*p)++; → 먼저, p번지의 값(*p)을 구하므로 b = 1

정답 : ③

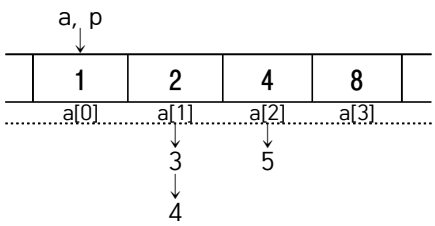
6. 다음 C 프로그램의 출력 값은? [2016년 국가 9급]

```
-----------------------------------------
 #include <stdio.h>
 int main()
 {
   int a[] = {1, 2, 4, 8};
   int *p = a;
   p[1] = 3;
   a[1] = 4;
   p[2] = 5;
   printf("%d, %d\n", a[1]+p[1], a[2]+p[2]);
   return 0;
 }
-----------------------------------------
```

① 5, 9
② 6, 9
③ 7, 9
④ 8, 10

♤ **배열과 포인터**

• 배열과 포인터는 서로 호환성이 있다.




```
// printf("%d, %d\n", a[1]+p[1], a[2]+p[2]);
```

• a[1] + p[1] = 4 + 4 = **8**
• a[2] + p[2] = 5 + 5 = **10**

정답 : ④

7. 다음 C 프로그램의 실행 결과는? [2016년 서울 9급]

```
#include<stdio.h>
int main()
{
  char *a[2] = {"Good morning", "C language"};
  printf("%s\n", a[0]+5);
  printf("%c\n", *(a[1]+6));
  return 0;
}
```

① Good morning
C-language

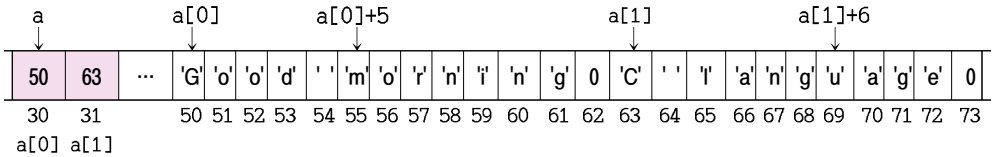
② morning
a

③ morning
g

④ morning
u

♤ 포인터 배열

• 메모리 구조는 다음과 같은 형태가 된다.(문자열 끝에 0은 null을 의미한다)
char *a[2] = {"Good morning", "C language"};



• 배열명 a는 배열의 시작주소 값을 가진다.

• printf("%s\n", a[0]+5);
→ 변환 기호가 %s이므로, a[0]+5가 가리키는 곳부터 해당 문자열 끝까지 출력
→ morning이 출력된다.

• printf("%c\n", *(a[1]+6));
→ 변환 기호가 %c이므로, a[1]+6가 가리키는 곳의 해당 문자 하나만 출력
→ u가 출력된다.

정답 : ④

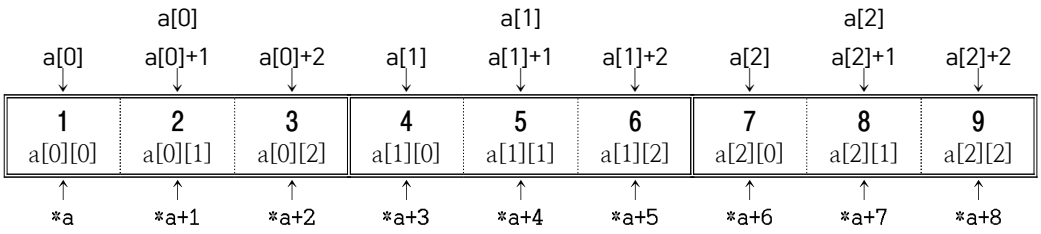
8. 다음 C 프로그램의 출력 값은? [2015년 지방 9급]

```
#include <stdio.h>
int main()
{
  int a[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
  int sum1, sum2;
  sum1 = *(*a + 1) + *(*a + 2);
  sum2 = *a[1] + *a[2];
  printf("%d, %d", sum1, sum2);
}
```

① 3, 5　　② 5, 5
③ 5, 11　　④ 11, 5

♤ **배열의 배열(2차원 배열)**

• 배열구조를 그림으로 나타내면 다음과 같다.



sum1 = *(*a + 1) + *(*a + 2) = 2 + 3 = **5**

sum2 = *a[1] + *a[2] = 4 + 7 = **11**

정답 : ③