

자료구조와 C

1. 배열 : 개념

● 배열 array

- 같은 자료형을 가진 자료들을 메모리에 연속으로 저장하여 만든 자료들의 그룹
- 인덱스 index
 - 배열의 요소를 구별하기 위해 사용하는 번호
 - C에서는 항상 0부터 시작
- 모든 자료형에 대해서 배열로 구성 가능
- 구성 형태에 따라 1차원 배열, 2차원 배열, 3차원 배열, ...

1. 배열 : 1차원 배열

● 1차원 배열 정의 형식

자료형	배열이름	[배열요소의 개수];
①	②	③

- ① 배열의 자료형을 선언한다. 배열 요소는 모두 자료형이 같아야 하고, 배열 요소의 자료형이 배열의 자료형이 된다.
- ② 변수 이름과 같은 규칙으로 정한다.
- ③ 대괄호([])를 사용해 배열 요소의 개수를 표시하는데, 배열 요소 개수가 배열 크기이다. 배열을 선언하면 메모리에 배열에 대한 공간이 할당되고 그 크기는 '자료형에 대한 메모리 할당 크기×배열 요소의 개수'이다.

그림 2-1 1차원 배열의 선언 형식

1. 배열 : 1차원 배열

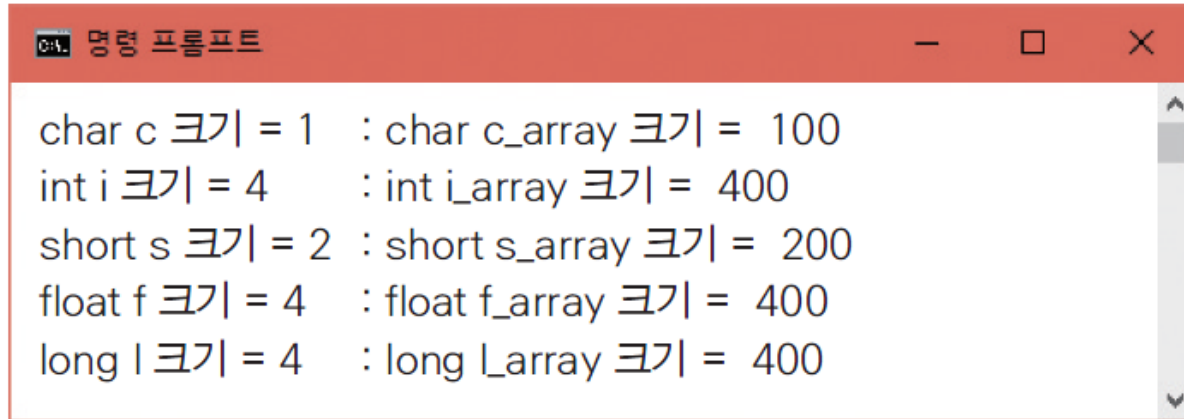
■ 예

표 2-1 여러 자료형의 배열 선언 예와 의미

배열 선언 예	의미	배열 요소	메모리 할당 크기
char c[100];	char형 배열 요소 100개로 구성된 배열 c	c[0] ~ c[99]	1byte x 100
int i[100];	int형 배열 요소 100개로 구성된 배열 i	i[0] ~ i[99]	4byte x 100
short s[100];	short형 배열 요소 100개로 구성된 배열 s	s[0] ~ s[99]	2byte x 100
long l[100];	long형 배열 요소 100개로 구성된 배열 l	l[0] ~ l[99]	4byte x 100

1. 배열 : 1차원 배열

- [예제 2-1] 자료형에 따른 메모리 할당 크기 확인하기



```
char c 크기 = 1 : char c_array 크기 = 100
int i 크기 = 4 : int i_array 크기 = 400
short s 크기 = 2 : short s_array 크기 = 200
float f 크기 = 4 : float f_array 크기 = 400
long l 크기 = 4 : long l_array 크기 = 400
```

- 배열의 정의와 메모리 할당 구조 예

```
int mid_score[40];
```



그림 2-2 배열 선언과 메모리 할당 구조 예 : 40명의 중간고사 점수

1. 배열 : 1차원 배열

● 1차원 배열의 초기화

■ 1차원 배열의 초기화 형식

자료형 배열이름[배열크기] = { 초깃값 리스트 };

그림 2-3 1차원 배열의 초기화

■ 1차원 배열의 초기화 예 1

`int A[5] = {1, 2, 3, 4, 5};`

또는

`int A[] = {1, 2, 3, 4, 5};`

또는

`int A[5];
A[0] = 1;
A[1] = 2;
A[2] = 3;
A[3] = 4;
A[4] = 5;`

배열의 모든 원소에 초깃값을 주면
배열 크기 생략 가능

(a) 1차원 배열의 초기화

	A[0]	A[1]	A[2]	A[3]	A[4]
A	1	2	3	4	5

(b) 메모리 할당 구조

그림 2-4 1차원 배열의 초기화 예 1

1. 배열 : 1차원 배열

■ 1차원 배열의 초기화 예 2

`int A[5] = {1, 2, 3};`

또는

`int A[5];
A[0] = 1;
A[1] = 2;
A[2] = 3;`



A[0]	A[1]	A[2]	A[3]	A[4]
1	2	3	0	0

(a) '초깃값의 개수 < 배열 크기'인 경우

`int A[3] = {1, 2, 3, 4, 5};`

또는

`int A[3];
A[0] = 1;
A[1] = 2;
A[2] = 3;
A[3] = 4;
A[4] = 5;`



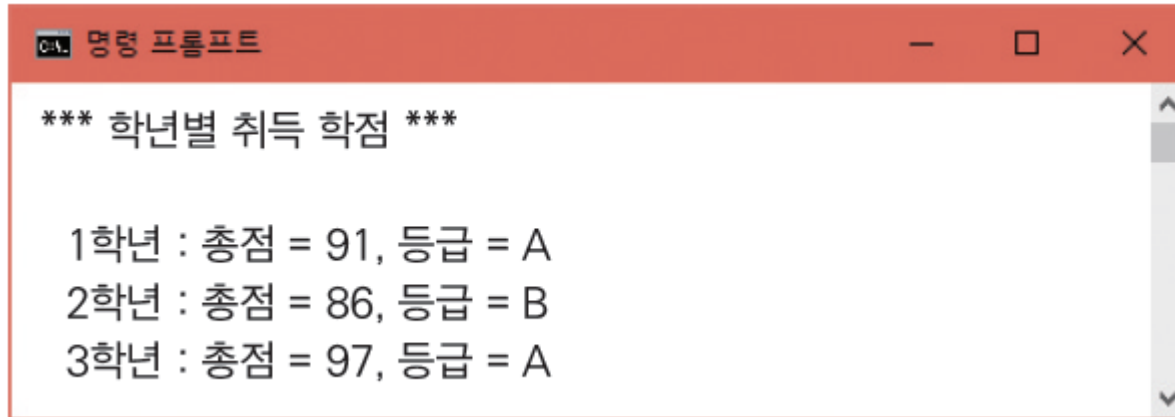
A[0]	A[1]	A[2]
1	2	3

(b) '초깃값의 개수 > 배열 크기'인 경우

그림 2-5 1차원 배열의 초기화 예 2

1. 배열 : 1차원 배열

- [예제 2-2] 학년별 취득 학점 입출력하기



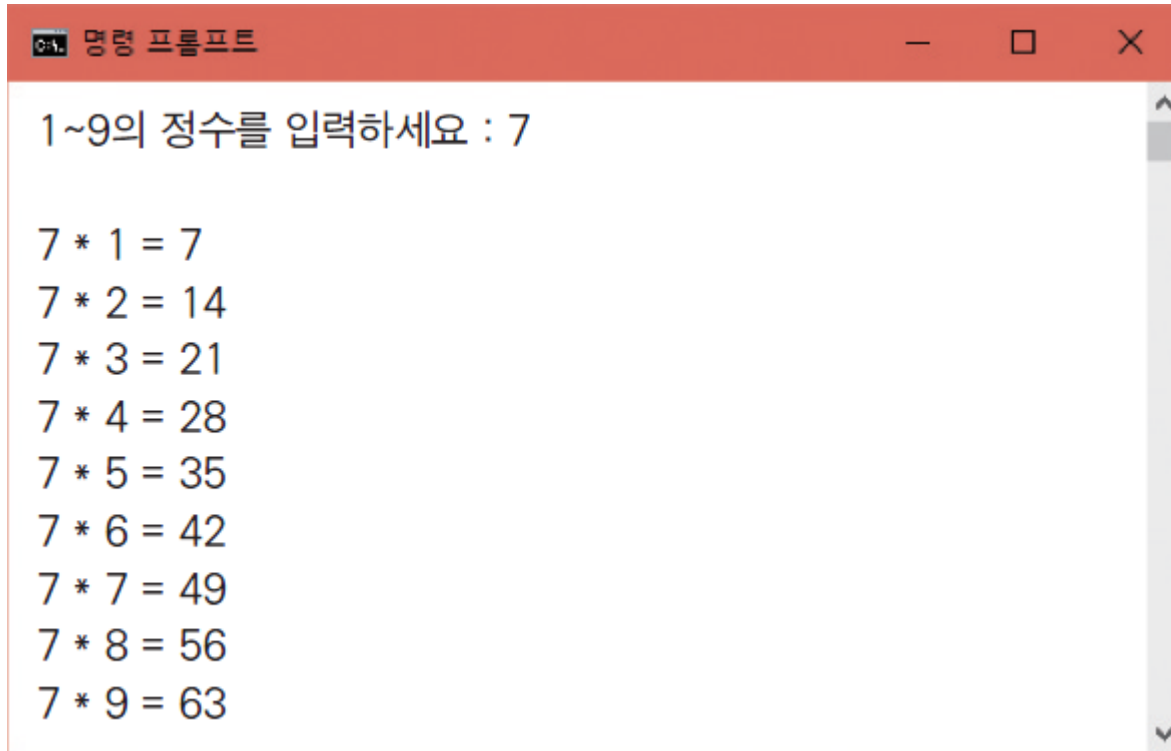
```
C:\> 명령 프롬프트

*** 학년별 취득 학점 ***

1학년 : 총점 = 91, 등급 = A
2학년 : 총점 = 86, 등급 = B
3학년 : 총점 = 97, 등급 = A
```


1. 배열 : 1차원 배열

- [예제 2-3] 입력한 숫자를 구구단으로 출력하기



A screenshot of a Windows Command Prompt window titled "명령 프롬프트". The window displays the following text:

```
1~9의 정수를 입력하세요 : 7  
  
7 * 1 = 7  
7 * 2 = 14  
7 * 3 = 21  
7 * 4 = 28  
7 * 5 = 35  
7 * 6 = 42  
7 * 7 = 49  
7 * 8 = 56  
7 * 9 = 63
```

1. 배열 : 1차원 배열

● 문자열

- 문자의 나열
- “와 ”사이에 표시
- 문자열을 저장하기 위해서는 문자열을 구성하는 문자들을 연속적으로 저장해야 하기 때문에 char형 배열을 사용
- 문자 배열의 초기화는 문자열 그대로 지정하거나 초깃값 문자 리스트 사용

1. 배열 : 1차원 배열

- 문자배열을 문자열 "String"으로 초기화하는 예

```
char s1[10] = "String";
```

	s1[0]	s1[1]	s1[2]	s1[3]	s1[4]	s1[5]	s1[6]	s1[7]	s1[8]	s1[9]
s1	S	t	r	i	n	g	\0			

(a) 문자열을 사용한 초기화

```
char s2[10] = { 'S', 't', 'r', 'i', 'n', 'g' };
```

	s2[0]	s2[1]	s2[2]	s2[3]	s2[4]	s2[5]	s2[6]	s2[7]	s2[8]	s2[9]
s2	S	t	r	i	n	g				

(b) 초깃값 문자 리스트를 사용한 초기화

그림 2-6 문자 배열의 선언과 메모리 할당 구조

- s1을 초깃값 문자 리스트를 사용 초기화

```
char s1[] = "String";
```

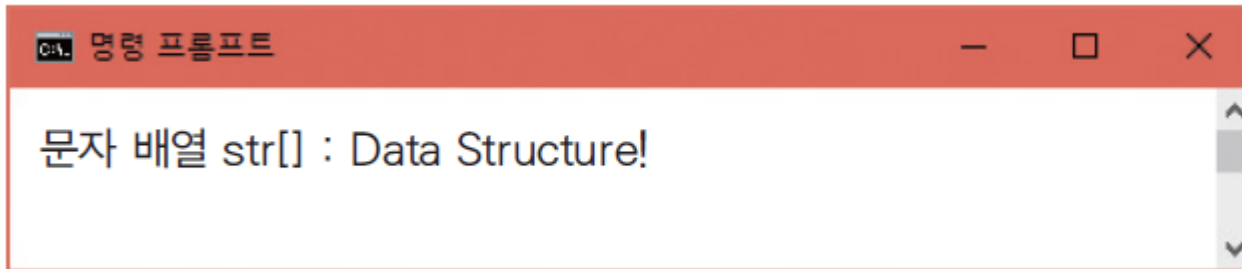


```
char s1[] = { 'S', 't', 'r', 'i', 'n', 'g', '\0' };
```

그림 2-7 문자열을 사용한 초기화 → 초깃값 문자 리스트를 사용한 초기화

1. 배열 : 1차원 배열

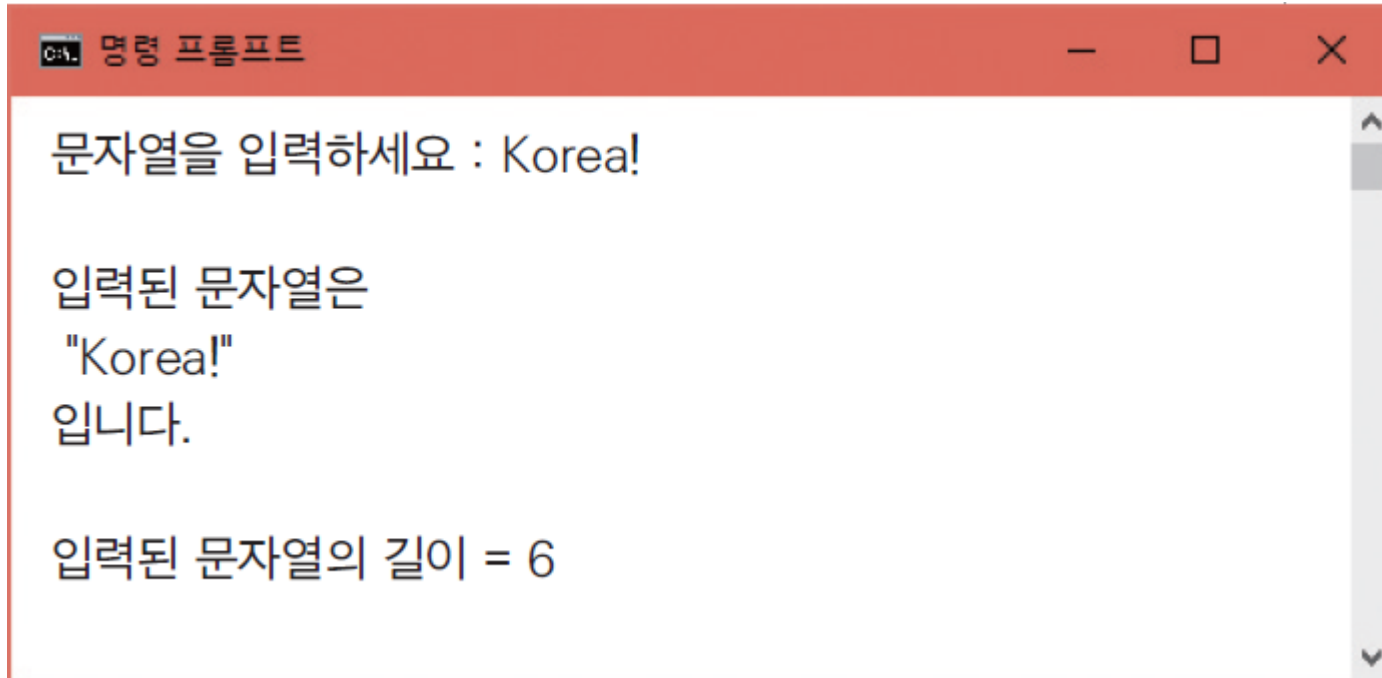
- [예제 2-4] 문자 배열에 문자열을 저장하고 출력하기



```
명령 프롬프트
문자 배열 str[] : Data Structure!
```

1. 배열 : 1차원 배열

- [예제 2-5] 입력한 문자열의 길이 계산하기



```
C:\> 명령 프롬프트

문자열을 입력하세요 : Korea!

입력된 문자열은
"Korea!"
입니다.

입력된 문자열의 길이 = 6
```

2. 포인터 : 개념

● 포인터 개념

- 변수의 메모리 주소값
- 포인터 변수
 - 주소값을 저장하는 특별한 변수
 - 포인터 변수가 어떤 변수의 주소를 저장하고 있다는 것은 포인터 변수가 그 변수를 가리키고 있다(포인팅하고 있다)는 의미
 - 포인터 변수를 이용하여, 연결된 주소의 변수 영역을 액세스 함
 - 포인터변수를 간단히 포인터라고 함

2. 포인터

■ 포인터 사용 예

```
int i;  
int *ptr = &i;
```

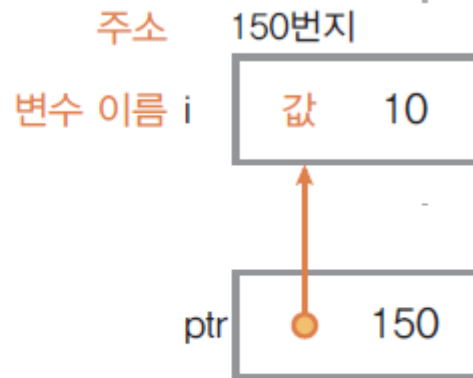


그림 2-19 포인터의 사용 예

2. 포인터 : 포인터 정의

● 포인터 정의

■ 포인터 정의 형식

<u>자료형</u>	<u>*포인터이름;</u>
①	②

① 포인터 자체의 자료형이 아니라 포인터에 저장할 주소에 있는 일반 변수의 자료형이다.

② 일반 변수 이름과 구별되도록 앞에 *를 붙여 포인터임을 나타낸다.

그림 2-20 포인터의 선언 형식

2. 포인터 : 포인터 정의

■ 포인터를 다양한 자료형으로 정의한 예

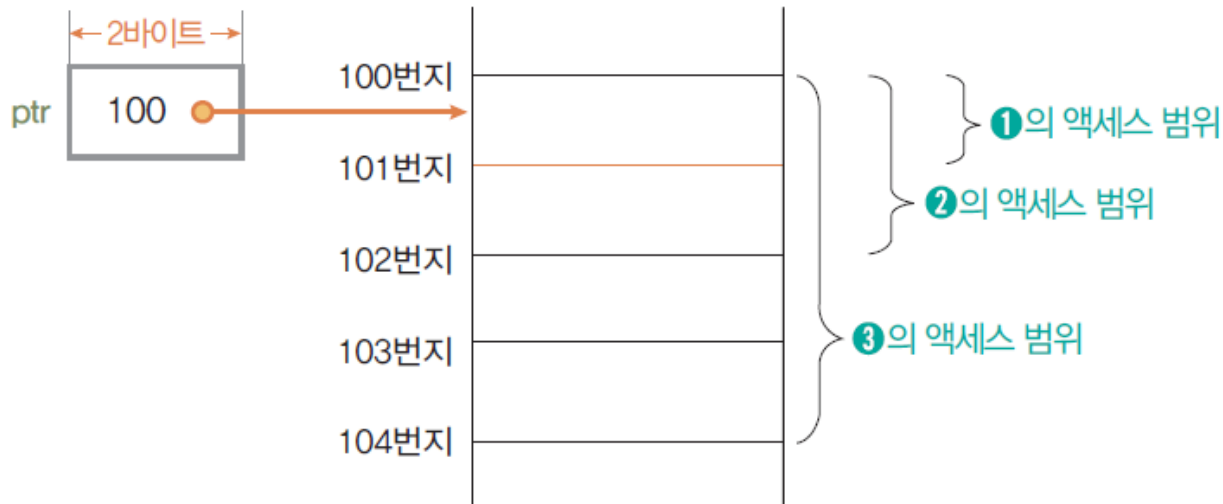
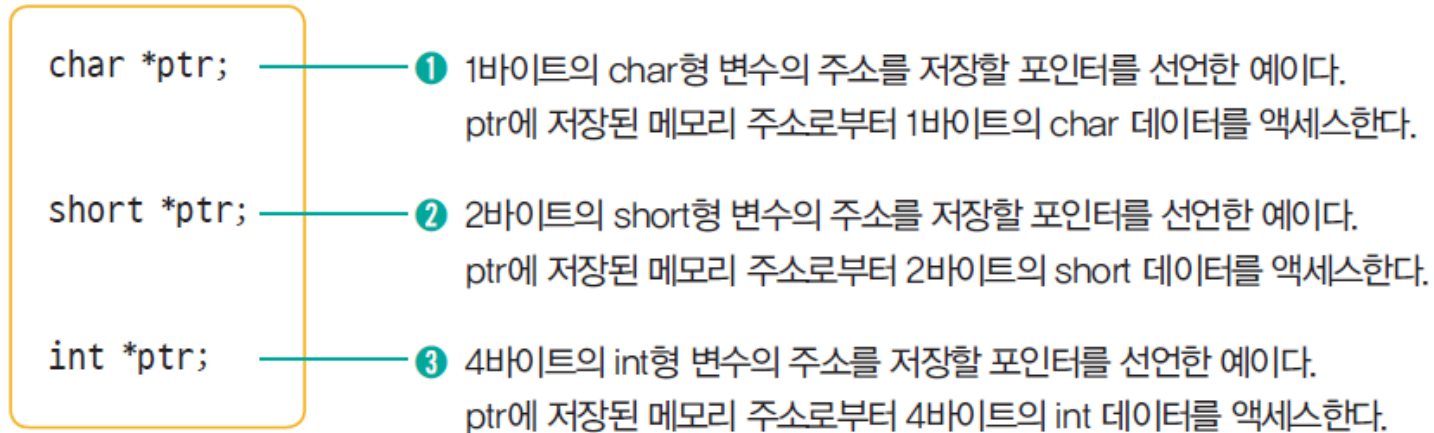


그림 2-21 포인터에서 선언한 자료형에 따른 메모리 액세스 범위

2. 포인터 : 포인터 연산

● 포인터 연산자

■ 주소 연산자 : $\&$

- 변수의 주소 얻기 위해 사용
- 형식

포인터 = $\&$ 변수;

그림 2-22 주소 연산자 사용 형식

- 예

(a)

```
int i = 10;  
int *ptr;
```

(c)

```
ptr = &i;
```

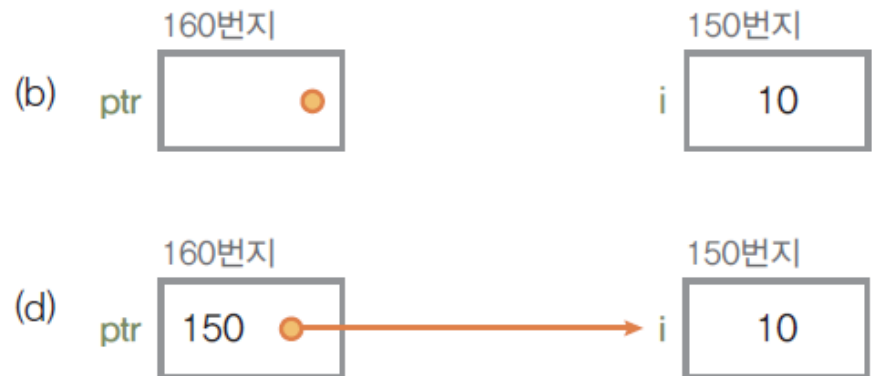


그림 2-23 포인터 선언과 사용 예

2. 포인터 : 포인터 연산

■ 참조 연산자 : *

- 포인터가 가리키는 기억공간 또는 가리키는 값을 의미하는 연산자
- 형식

① *포인터 = 값;

② 변수 = *포인터;

그림 2-24 참조 연산자 사용 형식

2. 포인터 : 포인터 연산

- 예

```
int i, j;
```

```
int *ptr;
```

```
ptr = &i; ❶ 주소 연산자를 사용하여 변수 i의 주소를 포인터 ptr에 할당한다. 포인터 ptr은 변수 i를 가리킨다.
```

```
*ptr = 10; ❷ 참조 연산자를 사용하여 포인터 ptr이 가리키는 영역에 값 10을 지정한다. 따라서 변수 i에는 10이 저장된다.
```

```
j = *ptr; ❸ 다시 참조 연산자를 사용하여 ptr이 가리키는 영역의 값을 변수 j에 지정한다. 따라서 ptr이 가리키는 변수 i의 값인 10을 변수 j에 저장한다.
```

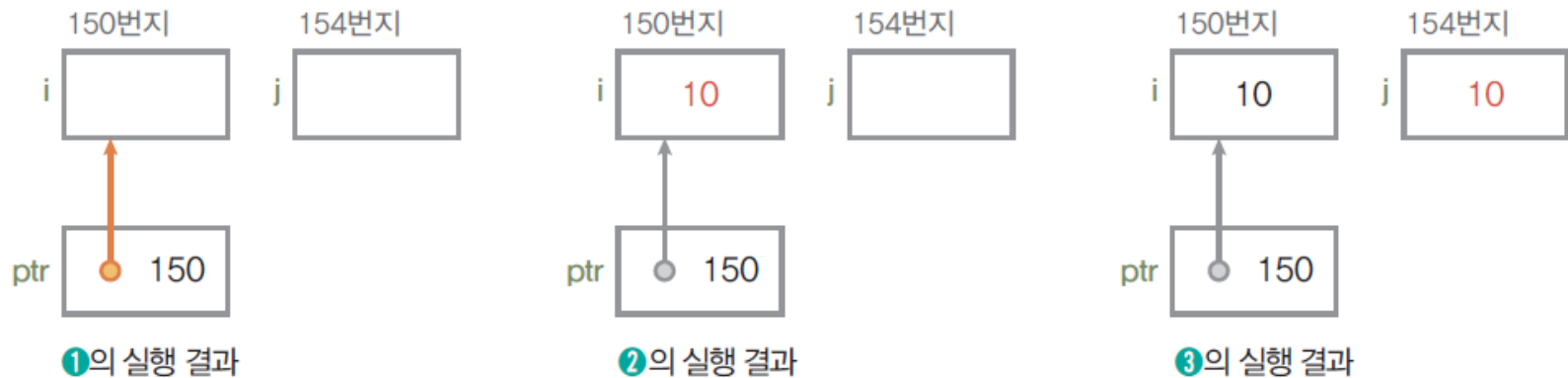


그림 2-25 포인터 연산자를 사용한 예

2. 포인터 : 포인터 연산

- [예제 2-8] 포인터 연산자를 이용해 변수 액세스하기

```
명령 프롬프트
i의 값 = 10
j의 값 = 20
i의 메모리 주소(&i) = 1245052
j의 메모리 주소(&j) = 1245048

<< ptr=&i 실행 >>
ptr의 메모리 주소(&ptr) = 1245044
ptr의 값(ptr) = 1245052
ptr의 참조값(*ptr) = 10

<< ptr=&j 실행 >>
ptr의 메모리 주소(&ptr) = 1245044
ptr의 값(ptr) = 1245048
ptr의 참조값(*ptr) = 20

<< i=*ptr 실행 >>
i의 값 = 20
```

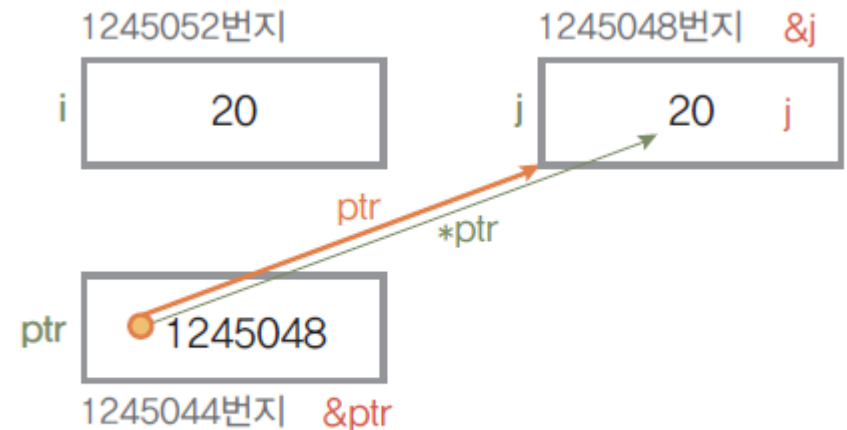


그림 2-26 [예제 2-8]을 실행한 결과

2. 포인터 : 포인터 연산

● 포인터의 초기화

- 일반 변수를 초기화하는 방법과 같음

- 포인터에 주소 지정 방법

- ❶ 주소 연산자를 사용하여 변수 주소를 지정

```
int i;  
int *ptr = &i;
```

- ❷ 동적 메모리를 할당하고 그 시작 주소를 포인터값으로 지정

```
char *ptr = (char *)malloc(100);
```

2. 포인터 : 포인터 연산

- ③ 문자형 포인터에 문자열의 시작 주소를 지정

```
char *ptr = "korea";
```

- ④ 배열 이름을 이용하여 배열 시작 주소를 지정

```
char A[100];  
char *ptr = A;
```

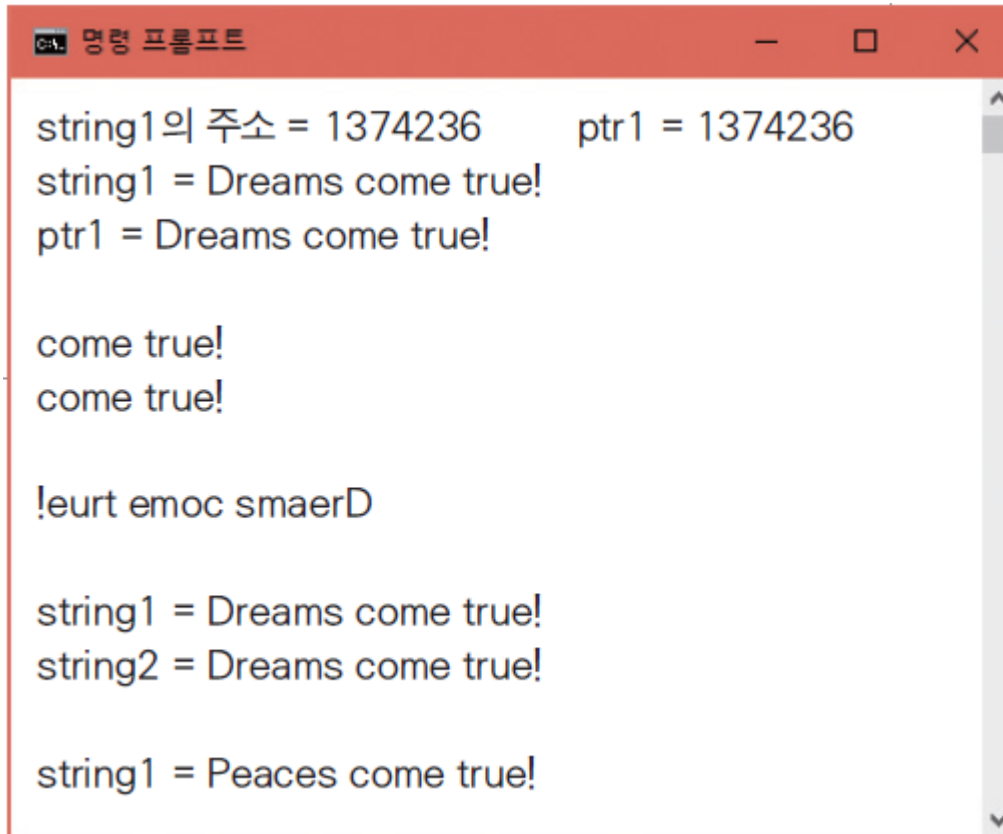
- ⑤ 배열의 첫 번째 요소의 주소를 이용하여 배열 시작 주소를 지정

```
char A[100];  
char *ptr = &A[0];
```

2. 포인터 : 포인터 연산

● 포인터와 문자 배열

- [예제 2-9] : 포인터를 이용해 문자열 처리하기



```
명령 프롬프트

string1의 주소 = 1374236    ptr1 = 1374236
string1 = Dreams come true!
ptr1 = Dreams come true!

come true!
come true!

!eurt emoc smaerD

string1 = Dreams come true!
string2 = Dreams come true!

string1 = Peaces come true!
```


3. 구조체 : 개념

● 구조체 개념

- 구조체도 배열처럼 여러 개의 데이터를 그룹으로 묶어서 하나의 자료형으로 선언하고 사용하는 자료형
 - 배열은 같은 자료형 만을 그룹으로 묶을 수 있지만, 구조체는 서로 다른 자료형을 그룹으로 묶을 수 있으므로 복잡한 자료 형태를 정의하는데 유용하게 사용됨
- 여러 자료형의 필드를 가지고 있는 레코드를 처리할 때 구조체 사용
- 필드, 레코드, 파일의 개념

파일

김선달	2014년 입사	3500
이몽룡	2015년 입사	3000
홍길동	2015년 입사	3200
향단이	2016년 입사	2900

← 레코드

↑ 필드

그림 2-30 필드, 레코드, 파일의 개념

3. 구조체 : 구조체 선언

● 구조체 선언

- 여러 자료형의 변수들을 그룹으로 묶어서 하나의 자료형으로 선언
- 구조체이름, 자료형, 데이터 항목으로 구성
 - 구조체의 이름 - 구조체로 정의하는 새로운 자료형의 이름
 - 항목 - 구조체를 구성하는 내부 변수들의 이름
 - 구조체의 항목은 배열의 각 배열요소에 해당
 - 배열요소는 모두 같은 자료형으로 되어 있으므로 배열요소에 대한 선언 없이 사용이 가능하지만, 구조체에서는 각 항목이 다른 자료형을 가질 수 있기 때문에 항목별로 자료형과 항목이름(변수이름)을 선언해야 한다.

3. 구조체 : 구조체 선언

■ 구조체형의 선언과 사용 형식

```
struct 구조체형이름 {  
    자료형 항목1;  
    자료형 항목2;  
    ...  
    자료형 항목n;  
};
```

(a) 선언 형식

```
struct 구조체이름 구조체변수이름;
```

(b) 사용 형식

그림 2-31 구조체형의 선언과 사용 형식

■ 구조체 사용 단계

- ❶ 구조체형 선언 : 내부 구조를 정의한다.
- ❷ 구조체 변수 정의 : 구조체형에 따른 변수를 정의한다.
- ❸ 구조체 변수의 사용 : 내부 항목에 데이터를 저장하고 사용한다.

3. 구조체 : 구조체 선언

■ 구조체 사용 예

```
struct employee {  
    char name[10];  
    int year;  
    int pay;  
};
```



그림 2-32 구조체형 employee 선언 예

```
struct employee Lee, Kim, Park;
```



그림 2-33 구조체형 employee에 대한 구조체 변수 선언 예

3. 구조체 : 구조체 선언

■ 구조체 변수의 정의 방법

표 2-2 구조체 변수의 선언 방법

방법	예
구조체형을 선언한 후에 구조체 변수 선언	<pre>struct employee { char name[10]; int year; int pay; }; struct employee Lee;</pre>
구조체형과 구조체 변수를 연결하여 선언	<pre>struct employee { char name[10]; int year; int pay; } Lee;</pre>
구조체형 이름을 생략하고 구조체 변수 이름만 선언	<pre>struct { char name[10]; int year; int pay; } Lee;</pre>

3. 구조체 : 구조체 변수의 초기화

● 구조체의 변수의 초기화

- 일반 변수 초기화와 마찬가지로 구조체 변수 초기화하려면 구조체 변수를 정의하면서 변수의 초깃값 지정
- 일반 변수는 값을 하나만 가지므로 초깃값도 하나지만, 구조체는 내부 항목이 여러 개일 수 있으므로 내부 항목의 자료형과 개수를 순서에 맞추어 초깃값 리스트로 지정하고 중괄호({ }) 사용

■ 예

```
struct employee { // 구조체형 선언
    char name[10];
    int year;
    int pay;
};
```

```
struct employee Lee = { "Ann", 2015, 4200 }; // 구조체 변수의 초기화
```

	name[10]								year	pay
Lee	A	n	n	\0					2015	4200

그림 2-34 구조체 변수를 초기화한 예

3. 구조체 : 데이터 항목의 참조

● 데이터 항목의 참조

- 구조체 연산자 사용 구조체 변수에 있는 각 데이터 항목 참조

표 2-3 구조체 연산자 종류

구조체 연산자	설명
점 연산자(.)	구조체 변수의 데이터 항목을 지정한다.
화살표 연산자(->)	구조체형 포인터에서 포인터가 가리키는 구조체 변수의 데이터 항목을 지정한다.

3. 구조체 : 데이터 항목의 참조

■ 점 연산자 : .

- 구조체 변수에 있는 데이터 항목을 개별적으로 지정할 때 사용
- 예

① struct employee Lee;

② Lee.name = ~~"Ann"~~;

③ Lee.year = 2015;

④ Lee.pay = 4200;



그림 2-35 점 연산자를 이용한 데이터 항목값 지정

3. 구조체 : 데이터 항목의 참조

- [예제 2-12] : 점 연산자를 이용해 데이터 항목 참조하기



3. 구조체 : 데이터 항목의 참조

■ 화살표 연산자 : ->

- 구조체 포인터 변수에서 포인터가 가리키는 구조체 변수의 데이터항목을 지정하기 위해서 화살표 연산자 사용
- 예

```
① struct employee Kim;  
② struct employee *Sptr = &Kim;  
③ Sptr->name = "susan";  
④ Sptr->year = 2014;  
⑤ Sptr->pay = 4300;
```

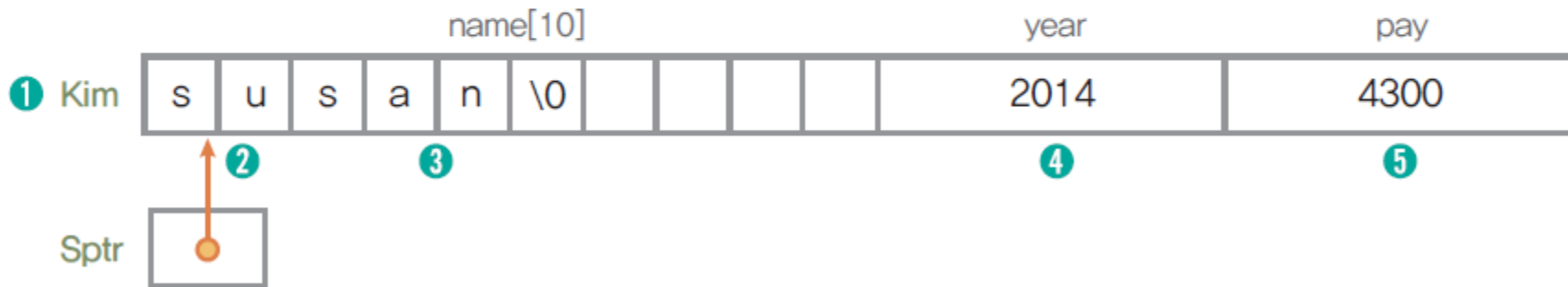


그림 2-36 화살표 연산자를 이용한 데이터 항목값 지정

3. 구조체 : 데이터 항목의 참조

- 구조체 포인터를 이용한 데이터 항목 지정 방법

```
Sptr->name = "susan";  
Sptr->year = 2014;  
Sptr->pay = 4300;
```

(a) 구조체 포인터의 화살표 연산자 사용



```
(*Sptr).name = "susan";  
(*Sptr).year = 2014;  
(*Sptr).pay = 4300;
```

(b) 구조체 포인터의 참조 연산자 사용

그림 2-37 구조체 포인터를 이용한 데이터 항목 지정 방법

괄호를 넣지 않으면 오류 발생

3. 구조체 : 데이터 항목의 참조

- [예제 2-13] : 화살표 연산자를 이용해 데이터 항목 참조하기



3. 구조체 : 구조체 연산

● 데이터 항목 참조 연산

- 점연산자와 화살표연산자 이용 구조체 데이터 항목 개별적으로 참조
- 예

```
struct employee Lee;  
struct employee *Sptr;  
Sptr = &Lee;  
Lee.year = 2015;  
Sptr->pay = 3000;  
Sptr->name = "Ann";
```

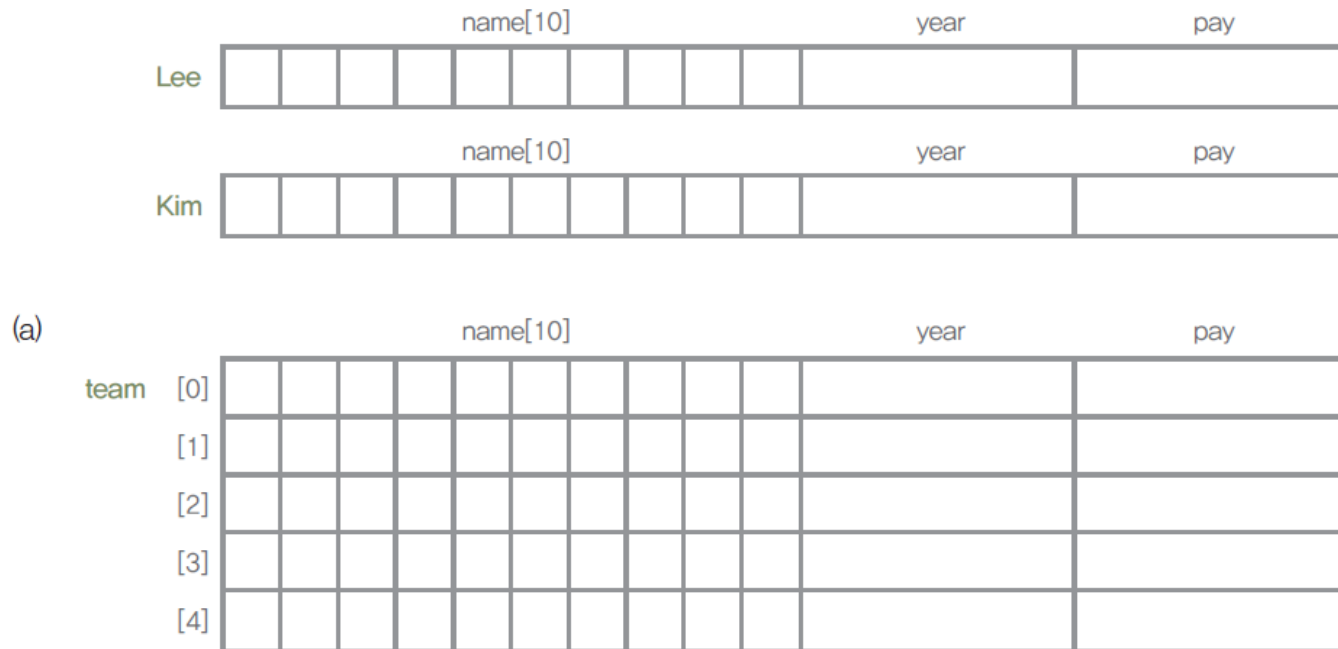
그림 2-38 데이터 항목 참조 연산 예

3. 구조체 : 구조체 연산

● 구조체 변수 복사 연산

- 같은 구조체에 있는 구조체 변수들끼리 내용을 한 번에 복사하는 연산

- 예 `struct employee Lee, Kim, team[5];`



(b)

```
Lee = Kim;  
Lee = team[2];  
team[2] = team[3];
```

그림 2-39 구조체 변수 복사 연산 예

3. 구조체 : 구조체 연산

- 구조체 변수의 주소 구하기 연산

- 포인터의 주소연산자 사용하여 구조체 변수의 주소 구하거나, 구조체 변수가 배열인 경우에는 배열의 특성에 따라 구조체 배열 변수의 이름에서 주소 구할 수 있음
- 예

```
struct employee Lee, team[5];  
struct employee *Sptr1, *Sptr2;  
  
Sptr1 = &Lee;  
Sptr2 = team;
```

그림 2-40 구조체 변수의 주소 구하기 연산 예

4. 재귀호출 : 재귀호출의 개념

● 재귀호출(순환호출)

- 자기 자신을 호출하여 순환 수행되는 것
- 함수 실행 특성에 따라 일반적인 호출방식보다 재귀호출방식을 사용하여 함수를 만들면 프로그램의 크기를 줄이고 간단하게 작성 가능
- 내가 나를 호출하는 것이므로 내 현재 작업을 처리하기 위해 같은 유형의 하위 작업이 필요
 - 하위 작업 : 현재 수행 중인 작업의 하위 단계, 즉 좀 더 작은 단위 작업
- 전제 문제를 한 번에 해결하기보다 같은 유형의 하위 작업으로 분할하여 작은 문제부터 해결하는 방법이 효율적인 경우 사용
- 베이스케이스^{base case}
 - 재귀호출하는 과정을 반복하다 보면, 한 번에 해결할 수 있을 정도로 분할된 작업 단위가 충분히 작아지는 단계

4. 재귀호출 : 재귀호출의 예

● 팩토리얼 함수

- n 에 대한 팩토리얼 함수는 1부터 n 까지 모든 자연수를 곱하는 연산

$$n! = n \times (n-1)!$$

$$(n-1)! = (n-1) \times (n-2)!$$

$$(n-2)! = (n-2) \times (n-3)!$$

...

$$2! = 2 \times 1!$$

$$1! = 1 \quad (\text{베이스케이스})$$

그림 2-43 $n!$ 연산

4. 재귀호출 : 재귀호출의 예

■ 예

```
long int fact(int n) {  
    if (n <= 1)  
        return (1);  
    else  
        return (n * fact(n - 1));  
}
```

(a) 재귀호출을 이용한 팩토리얼 함수

그림 2-44 팩토리얼 함수

```
long int fact(int n) {  
    int i, f = 1;  
    if (n <= 1)  
        return (1);  
    else {  
        for (i = n; i >= 0; i++)  
            f = f * i;  
        return f;  
    }  
}
```

(b) 반복문을 이용한 팩토리얼 함수

4. 재귀호출 : 재귀호출의 예

- factorial 함수에서 $n=4$ 일 때의 실행

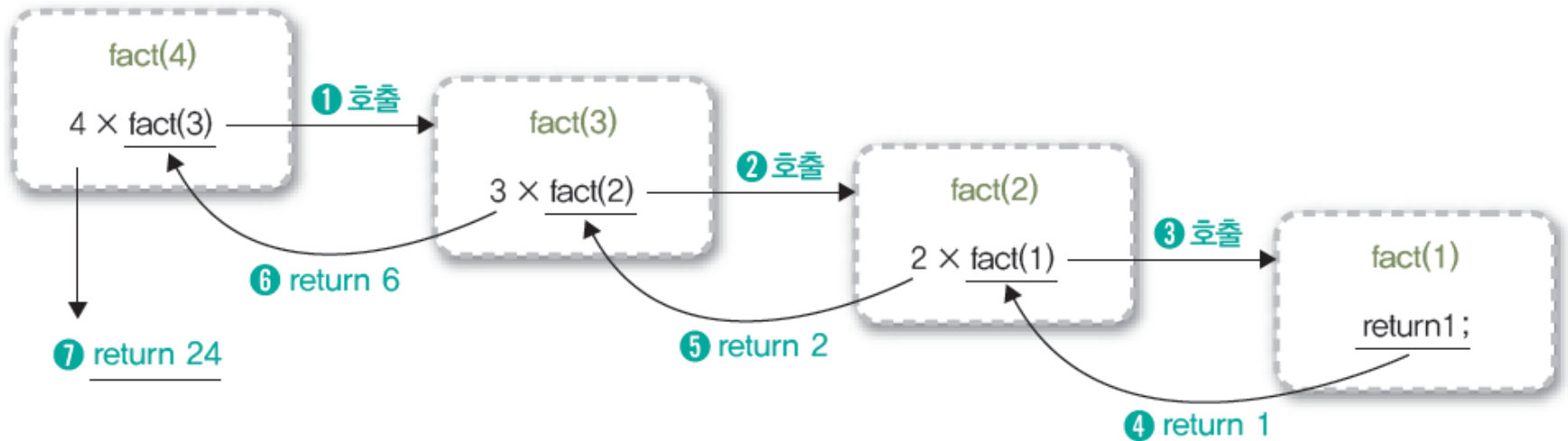
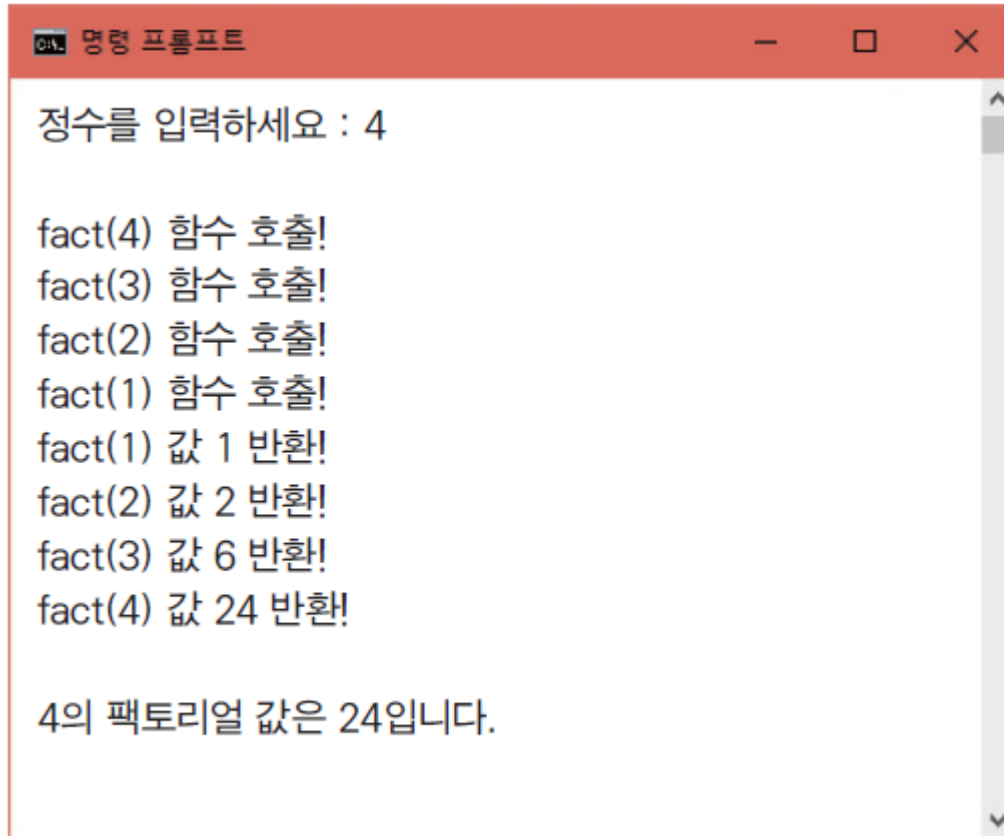


그림 2-45 $n=4$ 일 때 재귀호출과 반환

4. 재귀호출 : 재귀호출의 예

- [예제 2-14] : 재귀호출을 이용해 팩토리얼 값 구하기



```
명령 프롬프트

정수를 입력하세요 : 4

fact(4) 함수 호출!
fact(3) 함수 호출!
fact(2) 함수 호출!
fact(1) 함수 호출!
fact(1) 값 1 반환!
fact(2) 값 2 반환!
fact(3) 값 6 반환!
fact(4) 값 24 반환!

4의 팩토리얼 값은 24입니다.
```

5. 동적 메모리 할당

● malloc, free 함수

- 프로그램에 필요한 저장 공간
 - 프로그램을 작성할 때 변수나 배열의 정의를 통해서 확보 (정적 할당)
 - 프로그램 실행 중에 저장 공간을 할당(동적 할당)
 - 사용한 저장 공간은 재활용 위해 다시 반납
 - 메모리 동적 할당할 때 malloc 함수 사용
 - 반환할 때는 free 함수 사용

5. 동적 메모리 할당

● malloc, free 함수

- 프로그램에 필요한 저장 공간
 - malloc, free 함수 사용할 때 - stdlib.h 파일 include
 - 두 함수의 원형

```
void *malloc(unsigned int size);  
void free(void *p);
```

5. 동적 메모리 할당

● malloc, free 함수

예제 16-1 동적 할당한 저장 공간을 사용하는 프로그램

```
1. #include <stdio.h>
2. #include <stdlib.h>           // malloc, free 함수 사용을 위한 헤더 파일
3.
4. int main(void)
5. {
6.     int *pi;                  // 동적 할당 영역을 연결할 포인터 선언
7.     double *pd;
8.
9.     pi = (int *) malloc(sizeof(int)); // 메모리 동적 할당 후 포인터 연결
10.    if(pi == NULL)             // 동적 할당에 실패하면 NULL 포인터 반환
11.    {
12.        printf("메모리가 부족합니다.\n"); // 예외 상황 메시지 출력
13.        exit(1);                // 프로그램 종료
14.    }
15.    pd = (double *) malloc(sizeof(double));
16.
```

5. 동적 메모리 할당

● malloc, free 함수

```
17.    *pi = 10;                                // 포인터로 동적 할당 영역 사용
18.    *pd = 3.4;
19.
20.    printf("정수형으로 사용 : %d\n", *pi);      // 동적 할당 영역에 저장된 값 출력
21.    printf("실수형으로 사용 : %lf\n", *pd);
22.
23.    free(pi);                                  // 동적 할당 영역 반환
24.    free(pd);
25.
26.    return 0;
27. }
```

실행
결과

정수형으로 사용 : 10

실수형으로 사용 : 3.4

5. 동적 메모리 할당

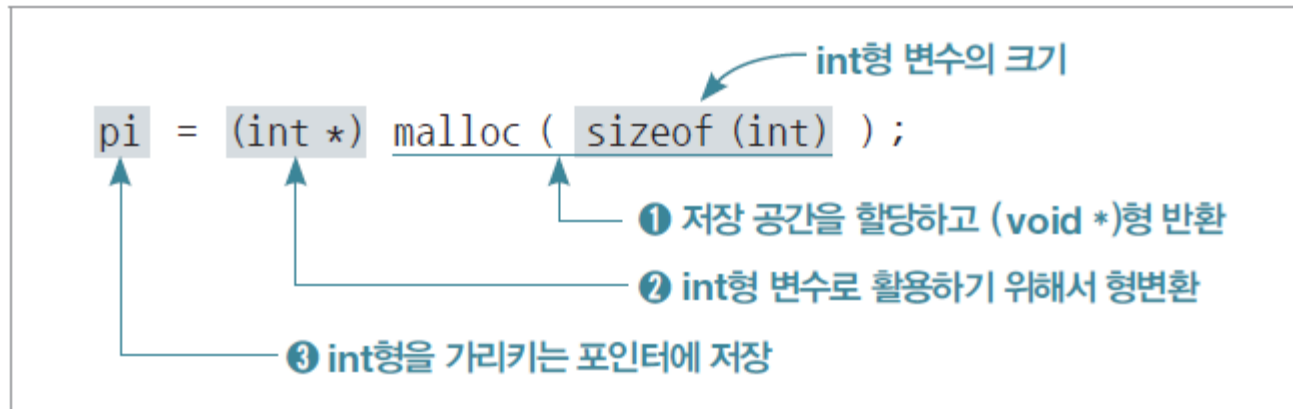
● malloc, free 함수

- 동적 할당한 저장공간을 사용하는 프로그램 풀이
 - 9행과 15행 - 메모리 동적 할당
 - int형 변수로 사용하기 위해서는 4바이트, double형 변수로 사용하기 위해서는 8바이트 할당
 - » 필요한 바이트 수 malloc 함수의 인수로 줄 것
 - » 각 자료형에 대한 크기를 계산하여 주는 것이 좋음
 - » 컴파일러에 따라 int형 변수의 크기가 다르더라도 프로그램 수정 필요 없음
 - » 주어진 인수의 바이트 크기 만큼 메모리에서 연속된 저장 공간 할당 후 그 시작 주소 반환

5. 동적 메모리 할당

● malloc, free 함수

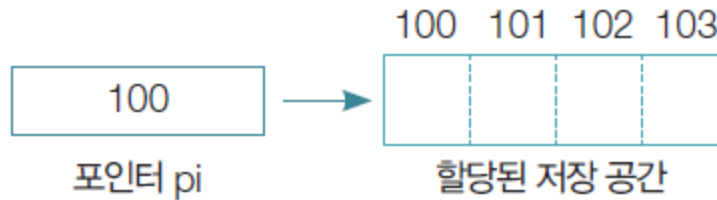
- 동적 할당한 저장공간을 사용하는 프로그램 풀이
 - malloc 함수는 (void *)형 반환
 - 용도에 맞는 포인터형으로 형변환하여 사용
 - 9행 동적 할당한 저장 공간을 int형 변수로 쓰기 위함
 - int형 가리키는 포인터에 저장



5. 동적 메모리 할당

● malloc, free 함수

- 동적 할당한 저장공간을 사용하는 프로그램 풀이
 - 메모리 100번지부터 할당되었다 가정할 경우



- 반환값 포인터에 저장 후
 - 간접참조 연산 수행하여 가리키는 저장 공간 값을 저장하거나 출력 가능

```
*pi = 10;           // pi가 가리키는 저장 공간에 10 저장  
printf("%d", *pi);  // pi가 가리키는 저장 공간의 값 출력
```

5. 동적 메모리 할당

● malloc, free 함수

- 동적 할당한 저장공간을 사용하는 프로그램 풀이
 - 동적으로 메모리 사용할 때 항상 포인터 쓰므로 주의할 점
 - malloc 함수 반환값이 널 포인터인지 반드시 확인 후 사용
 - » 메모리 할당 함수는 원하는 크기의 공간 할당하지 못하면 0번지인 널 포인터(null pointer) 반환
 - » 널 포인터는 보통 NULL로 표기하는데 전처리 단계에서 0으로 바뀌므로 정수0과 같다고 생각
 - » 포인터의 특별한 상태를 나타내기 위해 사용하므로 간접참조 연산불가

5. 동적 메모리 할당

● malloc, free 함수

- 동적 할당한 저장공간을 사용하는 프로그램 풀이
 - 동적으로 메모리 사용할 때 항상 포인터 쓰므로 주의할 점
 - malloc 함수 반환값이 널 포인터인지 반드시 확인 후 사용
 - » malloc 함수가 널 포인터 반환한 경우 그 값을 참조하면 실행 중 오류 메시지 표시하고 비정상 종료
 - » 프로그램이 실행될 때 메모리의 상태에 따라 달라짐
 - » 동적 할당 함수를 호출한 후에는 반드시 반환값을 검사하는 과정 필요

5. 동적 메모리 할당

● malloc, free 함수

- 동적 할당한 저장공간을 사용하는 프로그램 풀이
 - 10~14행
 - 메모리 할당하지 못한 경우
 - » 13행의 **exit** 함수가 프로그램 정상적으로 종료
 - » **exit** 함수는 **main** 함수뿐만 아니라 어떤 함수에서든 프로그램 바로 종료
 - » 예외 상황이 발생하여 프로그램 바로 종료하는 경우 인수로 1 주고 호출

5. 동적 메모리 할당

● malloc, free 함수

- 동적 할당한 저장공간을 사용하는 프로그램 풀이
 - 메모리 동적 할당할 때 사용이 끝난 저장 공간은 반환해야
- 자동 지역 변수의 저장 공간
 - 함수가 반환될 때 자동으로 회수
- 동적으로 할당한 저장공간
 - 함수가 반환된 후에도 자동으로 회수되지 않음
 - 반환되기 전에 free 함수로 직접 반환
 - » **main 함수가 끝날 때는 굳이 반환할 필요가 없음**
(프로그램 끝나면 동적 할당 부분 자동 반납)

5. 동적 메모리 할당

● malloc, free 함수

- 동적 할당한 저장공간을 사용하는 프로그램 풀이
 - 동적으로 할당한 저장공간
 - 그 외 다른 함수에서 사용하던 저장 공간은 불필요하면 반환
 - 메모리 반환코드 생략해도 당장 프로그램 실행에 영향 미치지 않아 반환 코드 생략 시
 - » **메모리 누수(memory leak) 발생 가능**
 - » **프로그램 의도치 않게 종료 가능**

5. 동적 메모리 할당

● malloc, free 함수

- 동적 할당한 저장공간을 사용하는 프로그램 풀이
 - 프로그램이 사용하는 메모리 영역은 기억부류 가짐
 - 프로그램은 실행될 때 일정한 메모리 영역 사용
 - 이 영역은 다시 몇 개의 영역으로 나뉘어 관리
 - » 기억부류(storage class)
 - » 구체적인 구분은 시스템에 따라 다름

5. 동적 메모리 할당

● malloc, free 함수

- 동적 할당한 저장공간을 사용하는 프로그램 풀이

- 기억 부류

- 프로그램이 올라가는 코드 영역

- 데이터가 저장되는 데이터 영역

» 지역 변수들이 할당되는 스택(stack) 영역

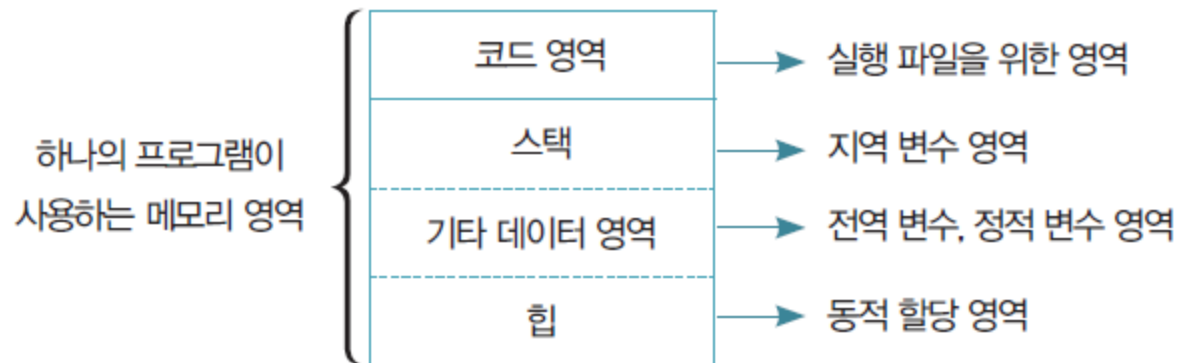
» 동적 할당되는 저장 공간은 힙(heap) 영역 사용

» 그 외 전역 변수나 정적 변수 위한 데이터 영역 존재

5. 동적 메모리 할당

● malloc, free 함수

- 동적 할당한 저장공간을 사용하는 프로그램 풀이
 - 기억 부류 구성도



5. 동적 메모리 할당

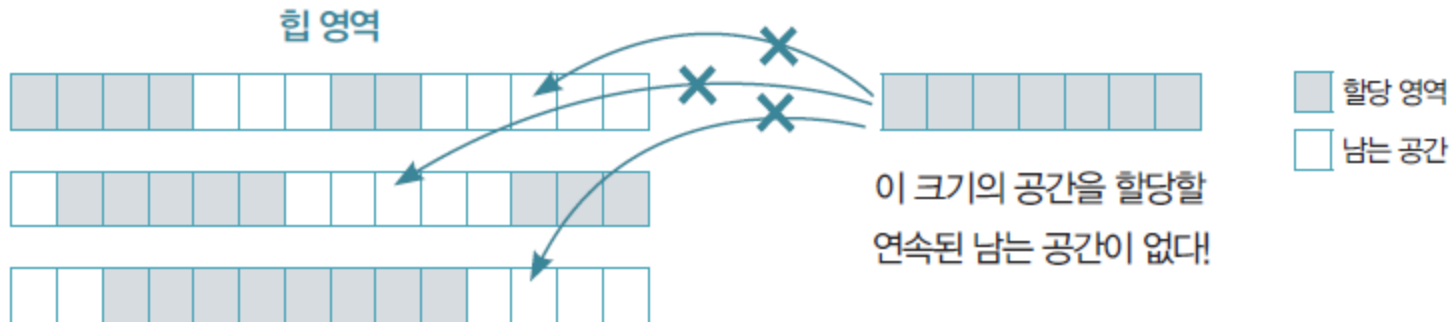
● malloc, free 함수

- 동적 할당한 저장공간을 사용하는 프로그램 풀이
 - 힙에 할당한 저장 공간
 - 지역 변수와 마찬가지로 쓰레기값 존재
 - 프로그램이 종료될 때까지 메모리에 존재
 - » 주소만 알면 특정 함수에 구매 받지 않고 어디서나 사용가능 지역 변수와 달리
 - 동적 할당된 저장 공간
 - 함수가 반환되어도 메모리 회수되지 않음
 - 메모리에 저장 공간이 넉넉히 남아 있어도 널 포인터 반환 가능

5. 동적 메모리 할당

● malloc, free 함수

- 동적 할당한 저장공간을 사용하는 프로그램 풀이
 - 힙 영역 - 메모리의 사용과 반환이 불규칙적
 - 사용 가능한 영역이 조각나서 흩어져 있을 수 있음
 - » 연속된 큰 저장 공간 요구하면 malloc 함수는 원하는 저장 공간 찾지 못하고 널 포인터 반환



- » 동적 할당 함수 호출한 후 반드시 반환값 검사하여 메모리의 할당 여부 확인해야