

Programming

Week 2. Review Basics

Joon-Woo Lee

School of Computer Science and Engineering
Chung-Ang University



목차

- 조건문
- 반복문
- 배열
- 함수

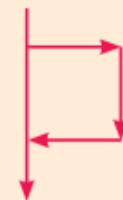
조건문

제어문 종류

- 조건 선택
 - 두개 또는 여러 개 중의 하나를 선택하는 구조
- 반복
 - 반복 몸체인 여러 문장을 여러 번 반복하는 구조
- 분기 처리
 - 정해진 장소로 이동하는 구조

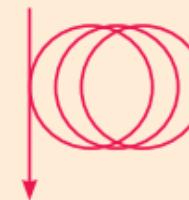
조건 선택

- 조건에 대한 선택 구문
- if
- if else
- if else if
- nested if
- switch



반복 순환

- 반복 조건에 따라 일정 영역의 구문 반복
- for
- while
- do while



분기처리

- 지정된 영역으로 실행을 이동하는 구문
- break
- continue
- goto
- return



조건에 따른 선택 if 문

평균평점 ≥ 3.5

대학 A는 평균평점이 3.5는 넘어야
장학금을 받을 수 있다.



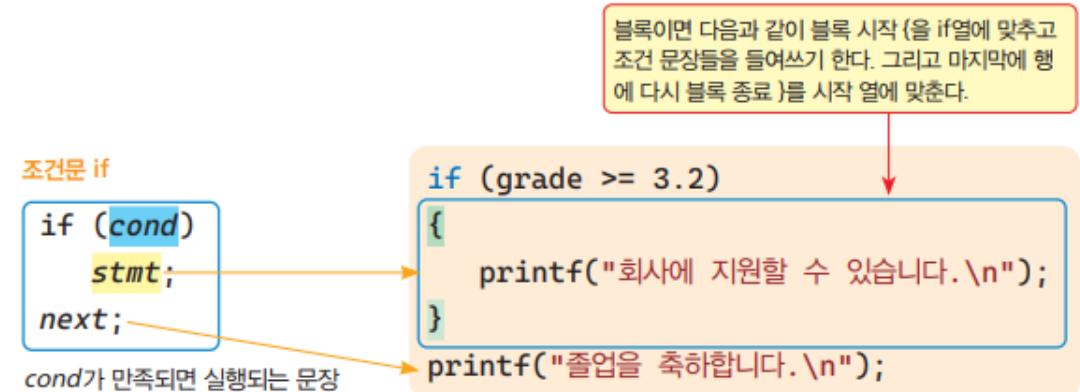
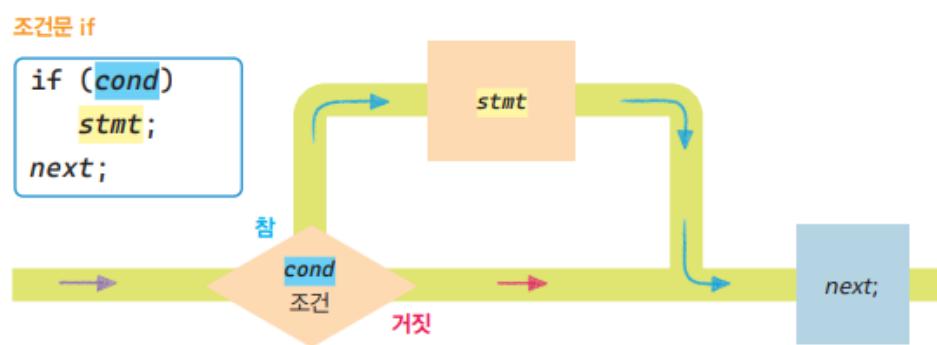
석차 $\leq 0.05 * \text{학생수}$

대학 B는 학과 석차가 상위 5%
이어야 장학금을 받을 수 있다고 한다.

조건 선택의 예	기준 변수	조건 표현의 의사코드
온도가 30도 이상이면 "폭염 주의"를 출력	온도 temperature	만일 (temperature $\geq 30)$ printf("폭염 주의");
낮은 혈압이 90이상이면 "고혈압 초기"로 진단	혈압 low_pressure	만일 (low_pressure $\geq 90)$ printf("고혈압 초기");
속도가 40km와 60km 사이이면 "적정 속도"라고 출력	속도 speed	만일 (40 \leq speed && speed $\leq 60)$ printf("적정 속도");
운전면허 필기시험에서 60점 이상이면 "합격", 아니면 "불합격" 출력	시험 성적 point	만일 (point $\geq 60)$ printf("면허시험 합격"); 아니면 printf("면허시험 불합격");
남성일 경우 체력 테스트에서 80점 이상이면 "합격"이 고, 아니면 "불합격", 여성이면 70점 이상이면 "합격", 아니면 "불합격"	성별 type 체력 점수 point	만일 남성이면 (type == 1) 만일 (point $\geq 80)$ printf("남성: 합격"); 아니면 printf("남성: 불합격"); 아니고 만일 여성이면 (type == 2) 만일 (point $\geq 70)$ printf("여성: 합격"); 아니면 printf("여성: 불합격");

조건에 따른 선택 if 문장

- if 문
 - 조건식이 0이 아니면(참) 문장을 실행
 - 0이면(거짓) 문장을 실행하지 않음



- 논리 오류와 문법 오류

```
if (grade >= 3.2);
printf("회사에 지원할 수 있습니다.\n");

printf("졸업을 축하합니다.\n");
```

```
if grade >= 3.2
printf("회사에 지원할 수 있습니다.\n");

printf("졸업을 축하합니다.\n");
```

현재 온도에 따른 폭염 주의 발령

실습예제 6-1 Prj01 01basicif.c 현재 온도에 따른 폭염 주의 발령 난이도: ★

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03
04 int main(void)
05 {
06     double temperature;
07
08     printf("현재 온도 입력: ");
09     scanf("%lf", &temperature);
10
11     if (temperature >= 30.0)
12     {
13         printf("폭염 주의보를 발령합니다.\n");
14         printf("건강에 유의하세요.\n");
15     }
16     printf("현재 온도는 섭씨 %.2f 입니다.\n", temperature);
17
18     return 0;
19 }
```

다음과 같이 블록 {의 시작을 조건식 오른쪽에 작성 하기도 함

```
if (temperature >= 32.0) {
    printf("폭염 주의보를 발령합니다.\n");
    printf("건강에 유의하세요.\n");
}
```

결과

현재 온도 입력: 29.3	현재 온도 입력: 34.678
현재 온도는 섭씨 29.30 입니다.	폭염 주의보를 발령합니다. 건강에 유의하세요.

현재 온도는 섭씨 34.68 입니다.

조건 만족 여부에 대한 선택 if else

- 조건을 만족하면 문장1을 실행
 - 조건을 만족하지 않으면 문장2를 실행하는 문장

조건문 if else

```
if (cond)
    stmt1;
else
    stmt2;
next;
```

```
if (n % 2 == 0)
    printf("짝수");
else
    printf("홀수");
printf("입니다.\n");
```

```
if (n % 2)
    printf("홀수");
else
    printf("짝수");
printf("입니다.\n");
```

조건식	설명	예	
(n != 0)	n이 0이 아니어야 참인 연산식이므로 연산식 (n)과 같음	<code>if (n % 2 != 0) printf("홀수"); else printf("짝수");</code>	<code>if (n % 2) printf("홀수"); else printf("짝수");</code>
(n)			
(n == 0)	n이 0이어야 참인 연산식이므로 연산식 (!n)과 같음	<code>if (n % 2 == 0) printf("짝수"); else printf("홀수");</code>	<code>if (!(n % 2)) printf("짝수"); else printf("홀수");</code>
(!n)			

조건 if else로 짝수와 홀수 판정

실습예제 6-2 Prj02 02evenifelse.c 조건 if else로 짝수와 홀수 판정 난이도: ★

```

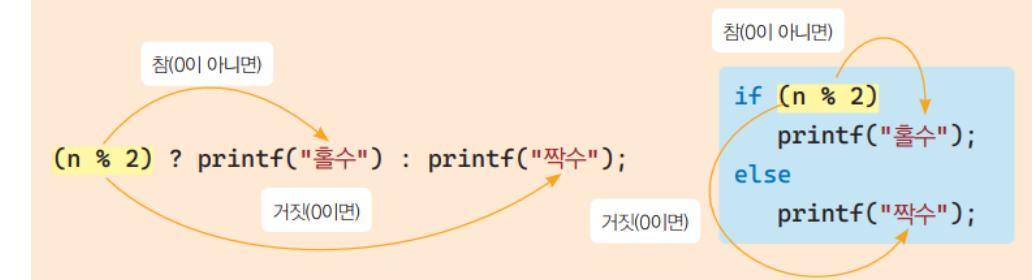
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int n;
07     printf("정수 입력: ");
08     scanf("%d", &n);
09
10     if (n % 2) // if (n % 2 != 0)
11     {
12         printf("홀수\n");
13     }
14     else
15     {
16         printf("짝수\n");
17     }
18
19 //조건연산자 이용
20 (n % 2) ? printf("홀수\n") : printf("짝수\n");
21
22     return 0;
23 }
```

결과

정수 입력: 5	정수 입력: 6
홀수	짝수
홀수	짝수

if else 문에서 조건식에 따른 실행문이 하나이더라도 블록을 구성해도 좋음
else 앞의 블록에 세미콜론 ; 을 붙이면 문법 오류가 발생하니 주의가 필요하다.

조건 문장 if는 이미 배운 조건연산자와 매우 유사해서 좀 더 편리하게 활용될 수 있다. 즉 다음 오른쪽의 if 문은 좌측의 조건연산자를 이용한 연산식으로 간단히 대체될 수 있다.



조건 if else 문에서 if 블록 이후, else 앞에 세미콜론 ;을 붙이면 문법 오류가 발생한다. 또한 else 바로 뒤에 세미콜론 ;을 붙이면 문법 오류는 없으나 논리 오류가 발생해 예상치 못한 결과가 발생하니 주의가 필요하다.

정상 실행	문법 오류	논리 오류
<pre>if (n % 2) { printf("홀수\n"); } else { printf("짝수\n"); }</pre>	<pre>if (n % 2) { printf("홀수\n"); };</pre>	<pre>if (n % 2) { printf("홀수\n"); }; else { printf("짝수\n"); }</pre> <p>else 문에 실행할 문장이 없이 종료되고, 이후 블록은 무조건 실행되는 논리 오류가 발생함</p>

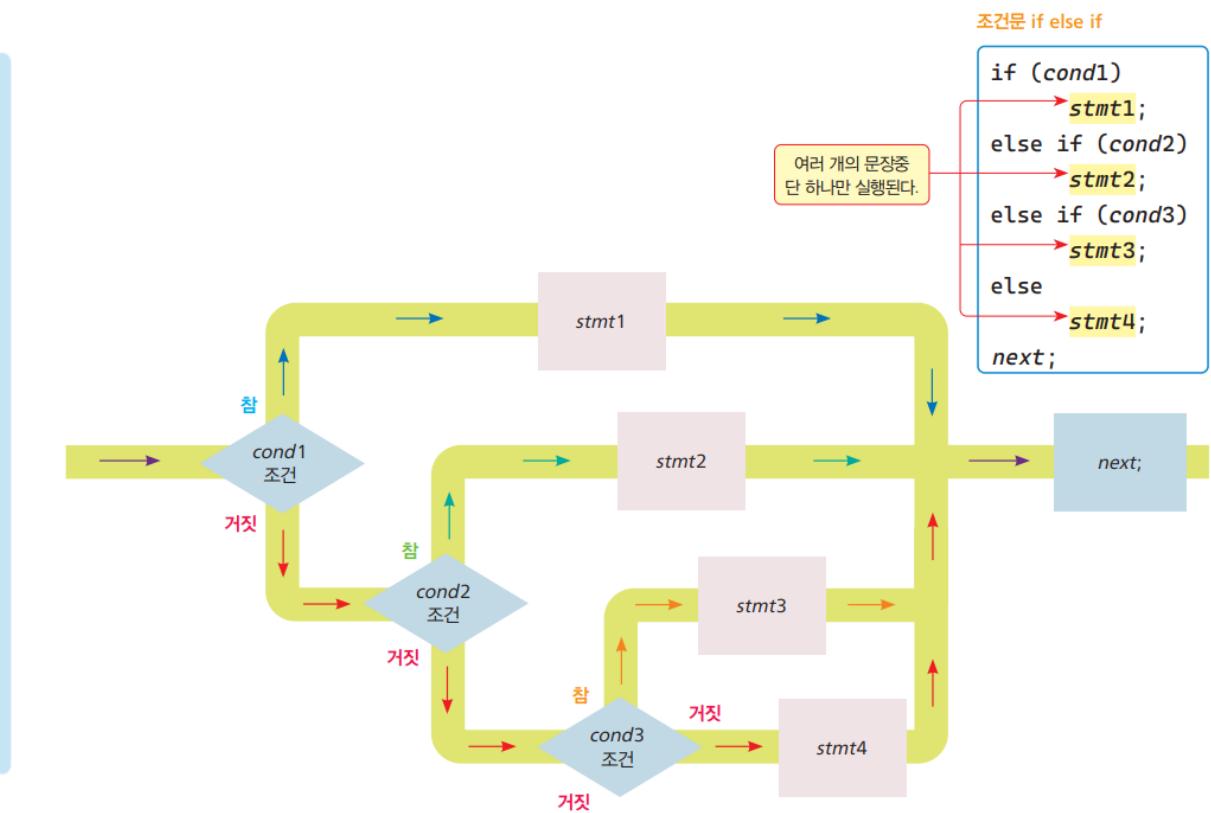
반복된 조건에 따른 선택 if else if

이 조건식은 첫 if의 조건식인 (point >= 90)이 만족되지 않고 체크되는 것이므로 결국 !(point >= 90) && (point >= 80)이므로 80 이상에서 90 미만인 조건 (90>point && point>=80)이 만족된다.

```
if (point >= 90)
    printf("A\n");
else if (point >= 80)
    printf("B\n");
else if (point >= 70)
    printf("C\n");
else if (point >= 60)
    printf("D\n");
else
    printf("F\n");
```

필요하면 이와 같이 블록 사용이 가능하다.

```
if (point >= 90)
{
    printf("A\n");
}
else if (point >= 80)
{
    printf("B\n");
}
else if (point >= 70)
{
    printf("C\n");
}
else if (point >= 60)
{
    printf("D\n");
}
else
{
    printf("F\n");
}
```



중첩된 if else 문으로 자동차 면허 합격 여부 판정

실습예제 6-4 Prj04 04nestedif.c 중첩된 if else 문으로 자동차 면허 합격 여부 판정 난이도: ★

```

01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int type, point;
07
08     printf("번호를 선택: 1(1종면허), 2(2종면허): ");
09     scanf("%d", &type);
10     printf("필기시험 점수 입력: ");
11     scanf("%d", &point);
12
13     if (type == 1)
14     {
15         if (point >= 70)
16             printf("1종면허 합격\n");
17         else
18             printf("1종면허 불합격\n");
19     }
20     else if (type == 2)
21     {
22         if (point >= 60)
23             printf("2종면허 합격\n");
24         else
25             printf("2종면허 불합격\n");
26     }
27
28     return 0;
29 }

```

결과

번호를 선택: 1(1종면허), 2(2종면허): 1 필기시험 점수 입력: 67 1종면허 불합격	번호를 선택: 1(1종면허), 2(2종면허): 1 필기시험 점수 입력: 77 1종면허 합격
번호를 선택: 1(1종면허), 2(2종면허): 2 필기시험 점수 입력: 58 2종면허 불합격	번호를 선택: 1(1종면허), 2(2종면허): 2 필기시험 점수 입력: 63 2종면허 합격



TIP

[코딩 주의] 조건식에서 등호 연산자 ==를 대입 연산자 =으로 잘못 코딩하는 경우

위 소스에서 다음과 같이 조건식 (type == 1)을 실수로 (type = 1)로 잘못 코딩하면 (type = 1)의 결과는 대입 값인 1이므로 if 조건을 항상 실행하는 논리 오류가 발생하니 주의가 필요하다. 초보자들에게 자주 발생하는 오류이니 조심하자.

```

if (type = 1)
{
    ...
}
else if (type == 2)
{
    ...
}
```



TIP

[코딩 주의] 조건식에서 등호 ==를 사용한 연산식에서 실수를 사용하는 경우의 문제

다음 코드의 결과는 '이상해요'가 출력된다. 변수 sum에는 float나 double의 문제로 실제 5.1이 아닌 5.1보다 조금 큰 실수가 저장된다. 그러므로 float나 double과 같은 실수를 관계 연산식이나 특히 등호 ==나 부등호 !=를 사용하는 경우 원하는 않는 결과가 발생할 수 있으니 가급적 사용하지 말자.

```

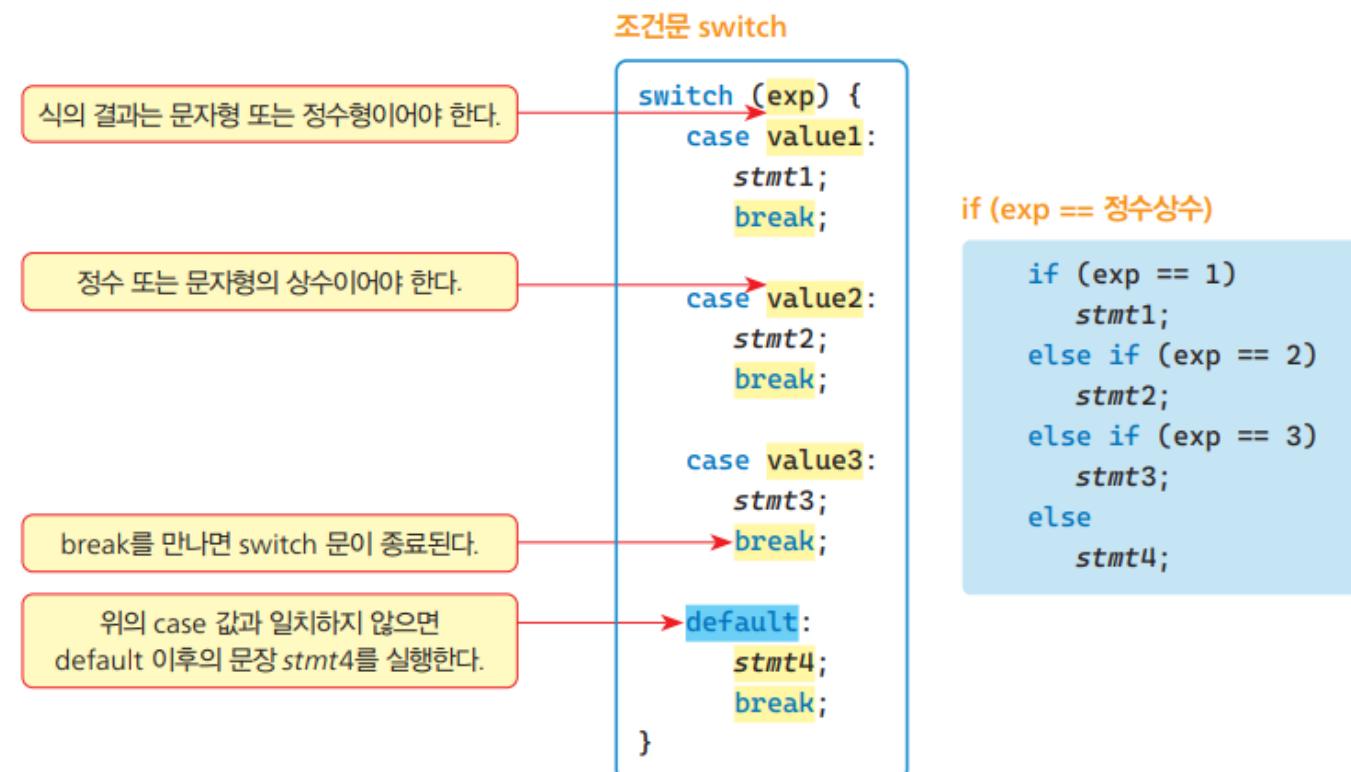
double a = 4.7, b = 0.4;
double sum = a + b;

if (sum == 5.1)
{
    printf("%s\n", "좋아요.");
}
else
{
    printf("%s\n", "이상해요.");
}
printf("%.20f ", sum);
```

이상해요.
5.10000000000000053291

switch 문장 개요

- switch 문
 - 연산식의 결과값에 따라 여러 경로 중에서 하나를 선택하는 선택하는 구문



switch로 두 실수의 사칙연산 수행

실습예제 6-6

Prj06 06arithswitch.c switch를 사용하여 두 실수의 사칙연산 수행 난이도 ★

```

01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03
04 int main(void)
05 {
06     double x, y, result;
07     int op;
08
09     printf("두 실수 입력: ");
10    scanf("%lf %lf", &x, &y);
11    printf("연산종류 번호선택 1(+), 2(-), 3(*), 4(/): ");
12    scanf("%d", &op);
13
14    switch (op)
15    {
16        case 1:
17            printf("%.2f + %.2f = %.2f\n", x, y, x + y);
18            break;
19        case 2:
20            printf("%.2f - %.2f = %.2f\n", x, y, x - y);
21            break;
22        case 3:
23            printf("%.2f * %.2f = %.2f\n", x, y, x * y);
24            break;
25        case 4:
26            printf("%.2f / %.2f = %.2f\n", x, y, x / y);
27            break;
28        default:
29            printf("번호를 잘못 선택했습니다. \n");
30            break; //생략 가능
31    }
32
33    return 0;
34 }
```

결과

두 실수 입력: 3.765 6.987	두 실수 입력: 4.82 3.987
연산종류 번호선택 1(+), 2(-), 3(*), 4(/): 1	연산종류 번호선택 1(+), 2(-), 3(*), 4(/): 2
3.77 + 6.99 = 10.75	4.82 - 3.99 = 0.83
두 실수 입력: 3.986 4.826	두 실수 입력: 87.354 6.98
연산종류 번호선택 1(+), 2(-), 3(*), 4(/): 3	연산종류 번호선택 1(+), 2(-), 3(*), 4(/): 4
3.99 * 4.83 = 19.24	87.35 / 6.98 = 12.51

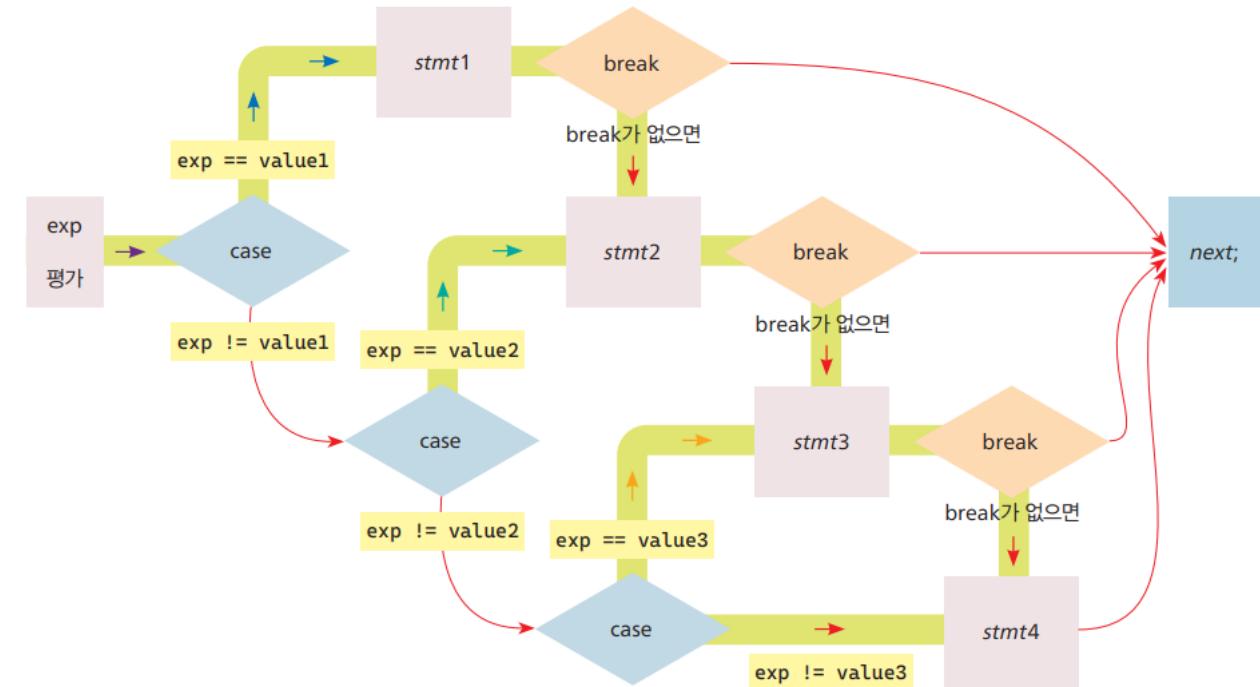


그림 6-20 switch 문의 제어흐름

default의 위치

- switch문 주의점
 - 연산식 결과는 정수형 또는 문자형
 - 각 case 뒤에 나오는 식은 상수식
 - 상수식에는 변수와 const 상수 사용 불가능
 - 리터럴 상수와 매크로 상수의 연산식은 사용 가능
- default
 - 선택적으로 없거나 하나이며
 - 어디에 위치해도 모든 case 처리를 하지 않은 경우 실행
 - 다른 case가 뒤에 있다면 break가 필요

실습예제 6-9 Prj09 09scoreswitch2.c 잘못된 점수도 고려하여 점수에 따른 성적 부여 난이도 ★★

```

01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int score;
07     printf("점수(0에서 100사이) 입력: ");
08     scanf("%d", &score);
09     if (score < 0 || score > 100)
10     {
11         printf("점수 입력이 잘못되었습니다.\n");
12         return 0;
13     }
14
15     switch (score / 10)
16     {
17         default:
18             printf("점수가 %d 점으로 성적이 %c 입니다.\n", score, 'F');
19             break;
20
21         case 10: case 9:
22             printf("점수가 %d 점으로 성적이 %c 입니다.\n", score, 'A');
23             break;
24         case 8:
25             printf("점수가 %d 점으로 성적이 %c 입니다.\n", score, 'B');
26             break;
27         case 7:
28             printf("점수가 %d 점으로 성적이 %c 입니다.\n", score, 'C');
29             break;
30         case 6:
31             printf("점수가 %d 점으로 성적이 %c 입니다.\n", score, 'D');
32             break;
33
34     }
35
36     return 0;
}

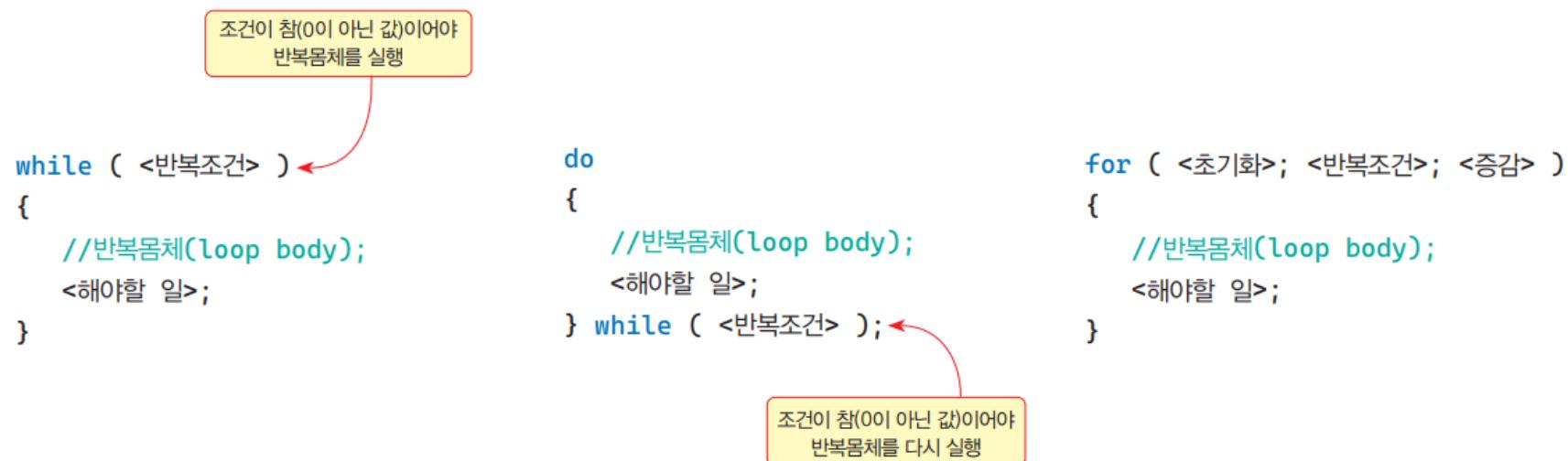
```

결과	점수(0에서 100사이) 입력: 101 점수 입력이 잘못되었습니다.	점수(0에서 100사이) 입력: 94 점수가 94 점으로 성적이 A 입니다.
	점수(0에서 100사이) 입력: 65 점수가 65 점으로 성적이 D 입니다.	점수(0에서 100사이) 입력: 55 점수가 55 점으로 성적이 F 입니다.

반복문

반복

- 반복
 - 순환 또는 루프(loop)라는 표현도 함께 사용
 - 반복몸체(repetition body)
 - 반복 조건을 만족하면 일정하게 반복되는 블록
- while, do while, for 세 가지 종류의 반복 구문



특정 문자열을 여러 번 반복해 출력

- 반복 제어 변수의 중요성
 - 반복 횟수에 따른 제어 변수의 값과 조건식의 결과를 살피는 것이 필요

실습예제 7-3 Prj03 03whilebasic.c 특정 문자열을 여러 번 반복해 출력 난이도: ★

```

01 #include <stdio.h>
02
03 int main(void)
04 {
05     int count = 1;
06
07     while (count <= 3)
08     {
09         printf("반복, 재미있네요!\n");
10         count++; // 반복문체로 제어변수 count를 1 증가시키는데, 결과값이 연산에 참여하지 않으므로 ++count도 가능하고, 결국 count += 1도 가능
11     };
12     printf("\n제어 변수 count => %d\n", count);
13
14     return 0;
15 }
  
```

반복 횟수 | 변수 count 값 | 조건식 | 조건식 평가 | 반복문체

1	1	conut <= 3 1 <= 3	while (1)	printf("C 언어 재미있네요!\n"); count++;
2	2	conut <= 3 2 <= 3	while (1)	printf("C 언어 재미있네요!\n"); count++;
3	3	conut <= 3 3 <= 3	while (1)	printf("C 언어 재미있네요!\n"); count++;
4	4	conut <= 3 4 <= 3	while (0) while 종료	실행되지 못함

결과

```

반복, 재미있네요!
반복, 재미있네요!
반복, 재미있네요!
  
```

제어 변수 count => 4

while 반복으로 표준입력 실수를 모두 더하기

실습예제 7-4 Prj04 04whilesum.c while 반복으로 표준입력 실수를 모두 더하기 난이도: ★

```

01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03
04 int main()
05 {
06     double number = 1, sum = 0;
07     while (number != 0.0)
08     {
09         printf("실수 입력 > ");
10         scanf("%lf", &number);
11         sum += number;
12     }
13
14     printf("합 = %.2f\n", sum);
15
16     return 0;
17 }
```

결과

```

실수 입력 > 5.9
실수 입력 > -3.4
실수 입력 > 5
실수 입력 > 0
합 = 7.50
```

논리 오류로 무한 반복 발생

```

int count = 1;
while (count <= 3)
    printf("C 언어 재미있네요!\n");
count++;
```

논리 오류로 무한 반복 발생

```

int count = 1;
while (count <= 3)
    printf("C 언어 재미있네요!\n");
count++;
```

블록 처리로 원하는 구문 처리

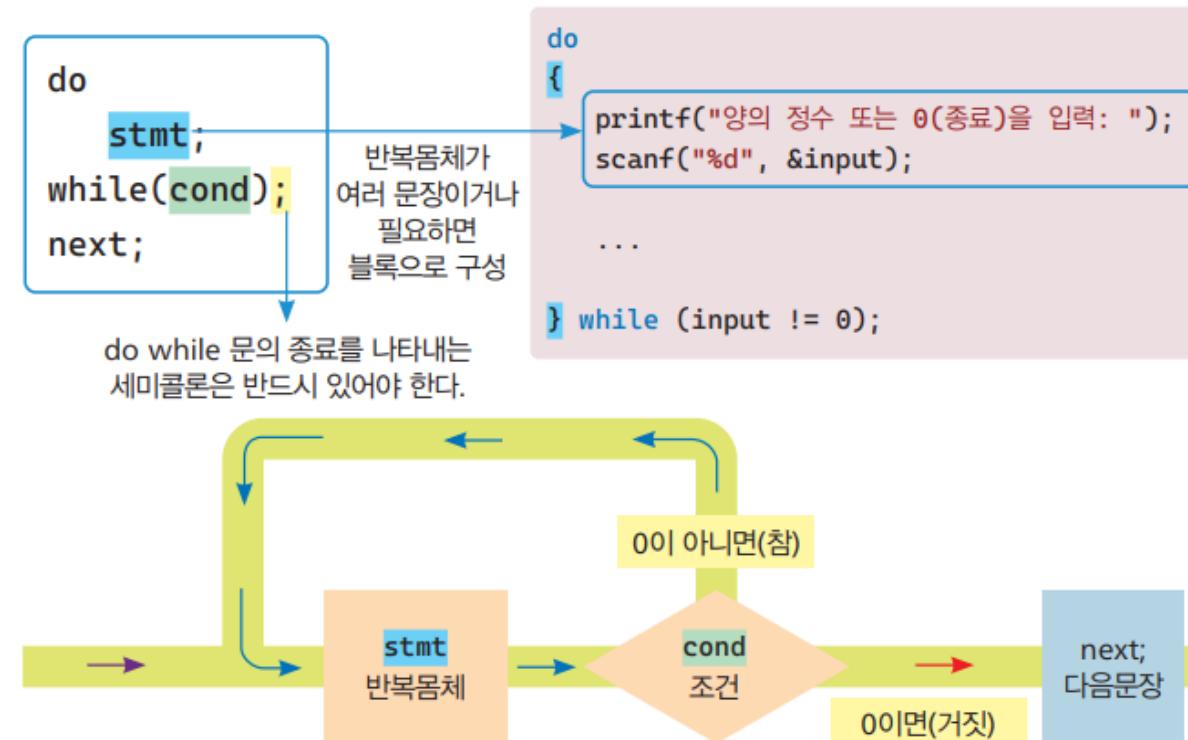
```

int count = 1;
while (count <= 3)
{
    printf("C 언어 재미있네요!\n");
    count++;
}
```

그림 7-12 while 블록의 중요성

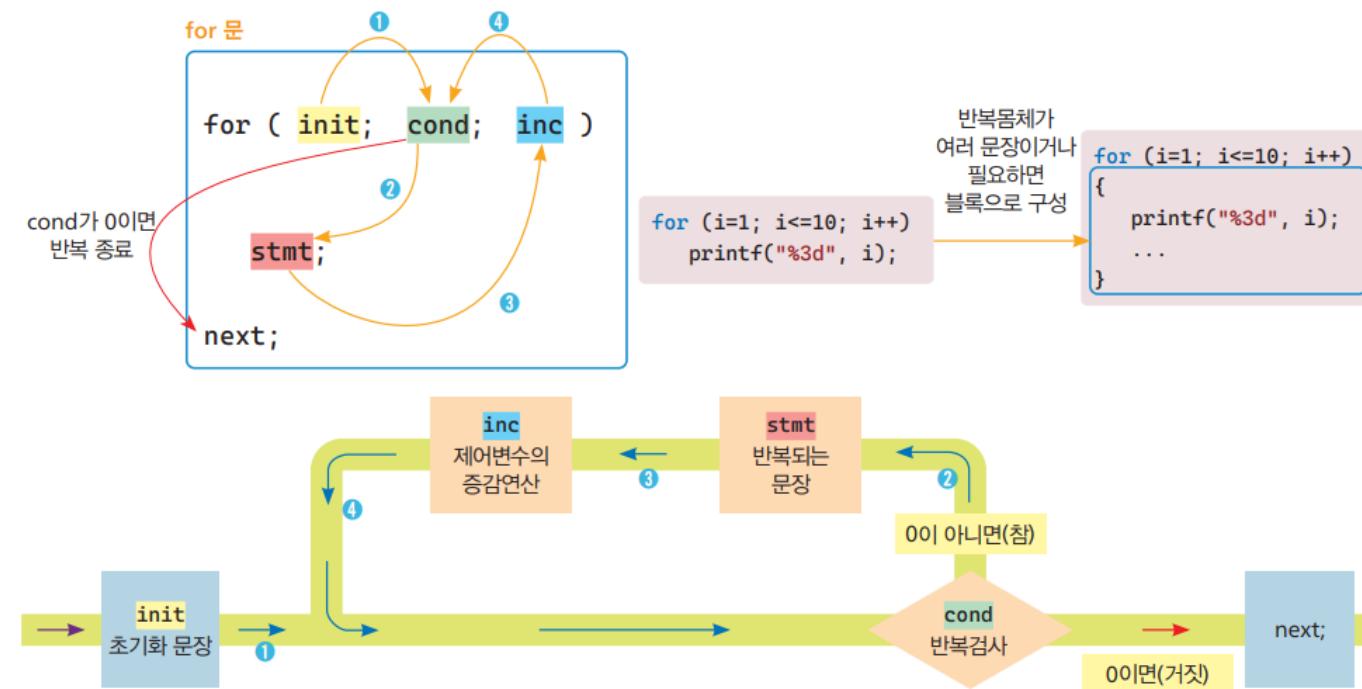
do while 문 구조와 제어흐름

- do while 문은 반복문체 수행 후에 반복 조건을 검사
 - while 문은 반복 전에 반복 조건을 평가
 - 반복 조건을 나중에 검사해야 하는 반복에 적합



for 문 구조와 제어 흐름

- 반복문 for (init; cond; inc) stmt;
 - init: 주로 초기화(initialization)
 - cond: 반복 조건을 검사
 - inc: 주로 반복을 결정하는 제어 변수의 증감(increment)을 수행



for 구문으로 일정 횟수 반복

- 2개의 세미콜론은 반드시 필요
- 반복조건 cond를 아예 제거하면 반복은 무한히 계속

실습예제 7-6 Prj06 06forbasic.c for 구문으로 일정 횟수 반복 난이도: ★

```

01 #include <stdio.h>
02 #define MAX 5
03
04 int main(void)
05 {
06     int i;
07     for (i = 1; i <= MAX; i++)
08         printf("반복 %d\n", i);
09
10     printf("\nfor 종료 이후 i => %d\n", i);
11
12     return 0;
13 }
```

조건식 $i \leq MAX$ 는 전처리 수행 후, MAX가 5로 대체되어 $i \leq 5$ 가 되며, i가 5보다 큰 6인 경우 조건식이 거짓이 되어 반복을 종료

증감의 $i++$ 는 반복문체인 8번 줄의 문장이 실행된 이후 실행

결과

반복 1
반복 2
반복 3
반복 4
반복 5

for 종료 이후 i => 6

초기화 문장은 단 한번만 실행된다.

```
for (int i = 1; i <= 10; i++)
    printf("%3d:", i);
```

이 부분은 $++i$, $i = i+1$, $i += 1$ 모두 가능하다.

```
int i = 0;
while (i <= 10)
{
    printf("%3d", i);
    i++;
}
```

21

반복 조건에서의 주의



TIP

반복 조건에서의 주의

반복 조건에서 등호나 부등호의 `==`나 `!=` 또는 대입연산자 `=`의 사용은 주의를 필요로 한다. 비교연산자 `==`와 `!=`에서 피연산자로 실수는 가급적 사용하지 않도록 하자. 예를 한 가지 들자면, 다음과 같이 0.0에서 0.1씩 증가시켜 1.0까지 10회를 반복하고자 하는 경우, 반복 조건을 `d != 1.0`으로 하면 실수 연산의 오차로 인해 조건식 `d != 1.00`이 항상 참인 결과로 반복이 무한히 계속될 수 있다. 다음의 왼쪽 소스는 무한히 반복되나 오른쪽과 같이 `d <= 1.0`으로 조건을 검사하면 0, 0.1, 0.2, …, 1.0까지 출력된다.

```
for (double d = 0.0; d != 1.0; d += 0.1)
    printf("%f ", d);
```

```
for (double d = 0.0; d <= 1.0; d += 0.1)
    printf("%f ", d);
```

그림 7-19 실수의 연산에서 `!=` 또는 `==`의 문제

또한 대입연산자 `=`과 등호 연산 `==`도 서로 혼동되지 않도록 유의하자. 다음 왼쪽 소스는 1 2 3 4 5가 출력되나 오른쪽 소스는 대입연산자인 `=`으로 잘못 사용하여 출력되는 것이 하나도 없다.

```
int i = 1;
while (!(i == 6))
{
    printf("%d ", i++);
}
```

```
int i = 1;
while (!(i = 6))
{
    printf("%d ", i++);
}
```

그림 7-20 `==`를 `=`로 잘못 사용

반복의 중단 break

- break 문장
 - 반복 내부에서 반복을 종료

실습예제 7-11 Prj11 11break.c 반복된 정수의 16진수 변환과 break로 종료 난이도: ★

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int input;
07     while (1) // while (1)에서 조건식 1이 항상 참이므로 무한반복
08     {
09         printf("양의 정수 또는 0[종료] 입력 후 [Enter] >> ");
10         scanf("%d", &input);
11         if (input == 0)
12             break;
13         printf("입력한 정수 %d: 16진수 %#x\n", input, input);
14     }
15     puts("종료");
16
17     return 0;
18 }
```

결과

```
양의 정수 또는 0[종료] 입력 후 [Enter] >> 15
입력한 정수 15: 16진수 0xf
양의 정수 또는 0[종료] 입력 후 [Enter] >> 16
입력한 정수 16: 16진수 0x10
양의 정수 또는 0[종료] 입력 후 [Enter] >> 0
종료
```

반복의 계속 continue

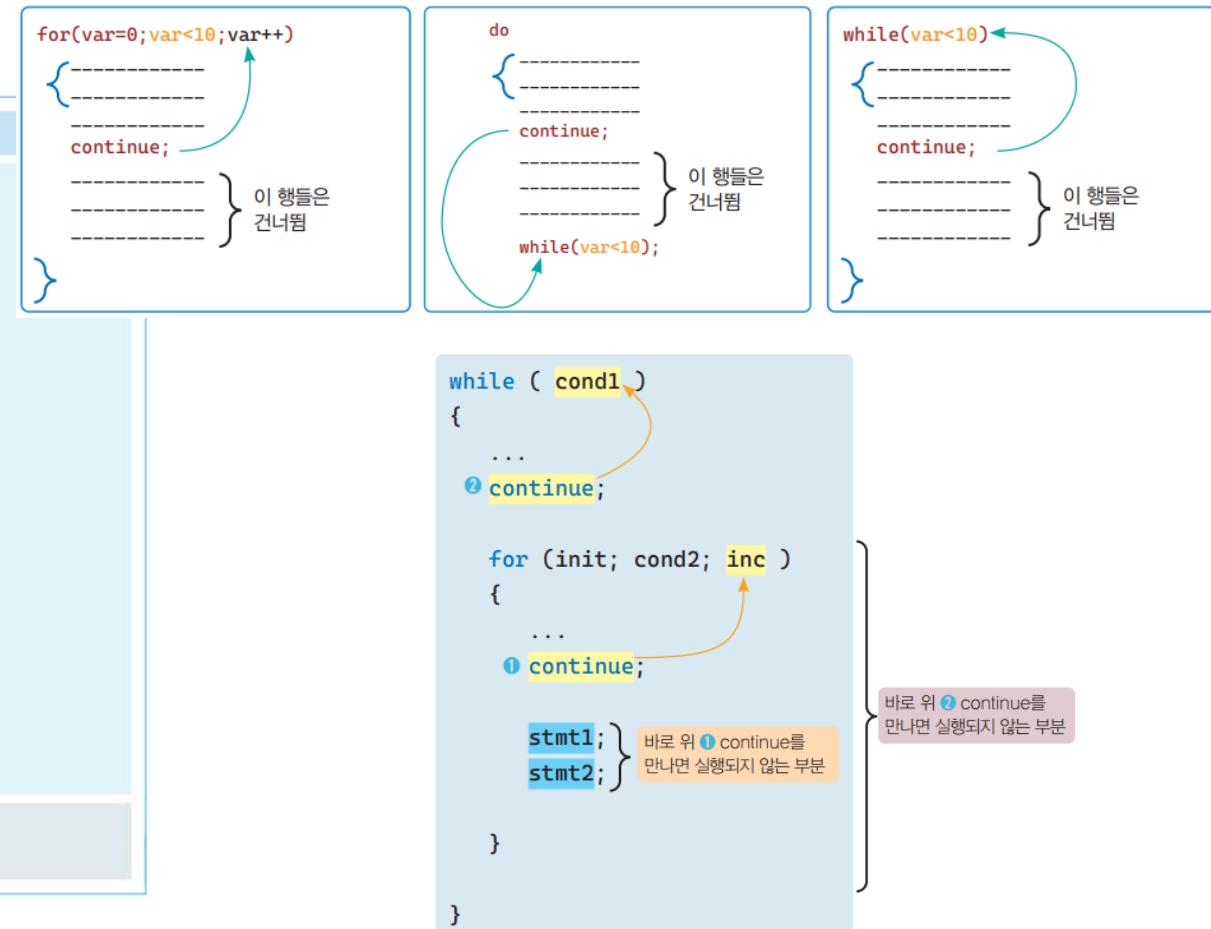
- continue 문장
 - continue는 자신이 속한 가장 근접한 반복에서 다음 반복을 실행

실습예제 7-12 Prj12 12continue.c 3으로 나누어지지 않는 정수 출력

```

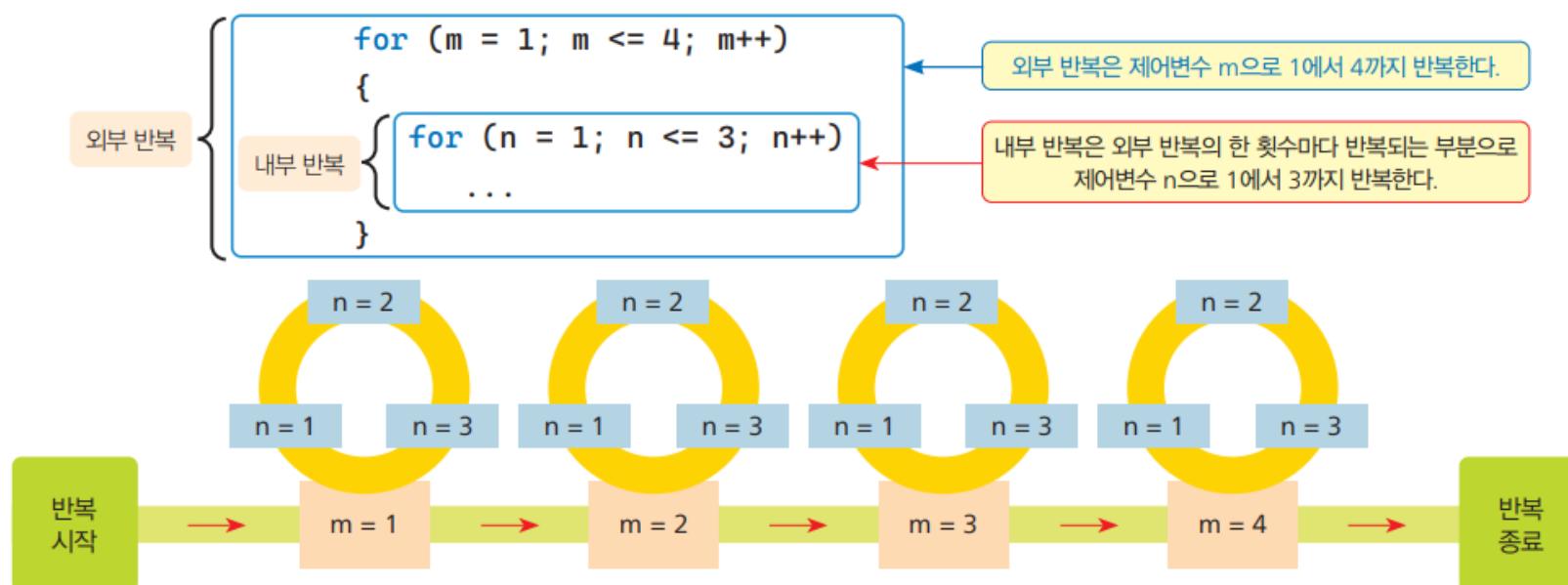
01 #include <stdio.h>
02
03 int main(void)
04 {
05     const int MAX = 15;
06
07     printf("1에서 %d까지 정수 중에서 3으로 나누어 떨어지지 않는 수\n", MAX);
08     for (int i = 1; i <= MAX; i++)
09     {
10         if (i % 3 == 0) // !(i % 3)
11             continue;
12         printf("%3d", i);
13     }
14     puts("");
15
16     return 0;
17 }
```

결과 1에서 15까지 정수 중에서 3으로 나누어 떨어지지 않는 수
1 2 4 5 7 8 10 11 13 14



중첩된 for 문

- 중첩된 반복문
 - for 문 내부에 for 문이 존재
 - 제어 변수는 m, n
 - 외부 for 문의 제어 변수는 m이며, 내부 for 문의 제어 변수는 n
 - 외부 반복에서 m은 1에서 4까지 반복
 - 각각의 m에 대해, 내부 반복에서 n이 1에서 3까지 반복



중첩 반복

- 내부 반복과 외부 반복에서 각각의 변수 값의 변화를 이해
 - 외부 반복에서 1에서 5까지,
 - 내부 반복에서 1에서 7까지 반복하면서
 - 각각의 변수 값을 출력하는 예제 프로그램

실습예제 7-15 Prj15 15nestedloop.c 내부 반복과 외부 반복에서 각각의 변수 값의 변화를 이해 난이도: ★

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int m, n;
06     for (m = 1; m <= 5; m++)
07     {
08         printf("m = %-2d\n", m);
09         for (n = 1; n <= 7; n++)
10             printf("n = %-3d", n);
11         puts("");
12     }
13
14     return 0;
15 }
```

외부 반복의 for 문으로 1에서 5까지 반복
내부 반복의 for 문으로 1에서 7까지 반복

결과

```
m = 1
n = 1  n = 2  n = 3  n = 4  n = 5  n = 6  n = 7
m = 2
n = 1  n = 2  n = 3  n = 4  n = 5  n = 6  n = 7
m = 3
n = 1  n = 2  n = 3  n = 4  n = 5  n = 6  n = 7
m = 4
n = 1  n = 2  n = 3  n = 4  n = 5  n = 6  n = 7
m = 5
n = 1  n = 2  n = 3  n = 4  n = 5  n = 6  n = 7
```

내부 반복이 외부 반복에 의존

- 외부 반복에서 변수 i는 1에서 5까지 반복
 - 내부 반복에서 제어 변수 j는 1에서 외부 반복의 제어 변수 i까지 반복

실습예제 7-16

Prj16 16triangle.c 별을 삼각형 모양으로 출력 난이도: ★★

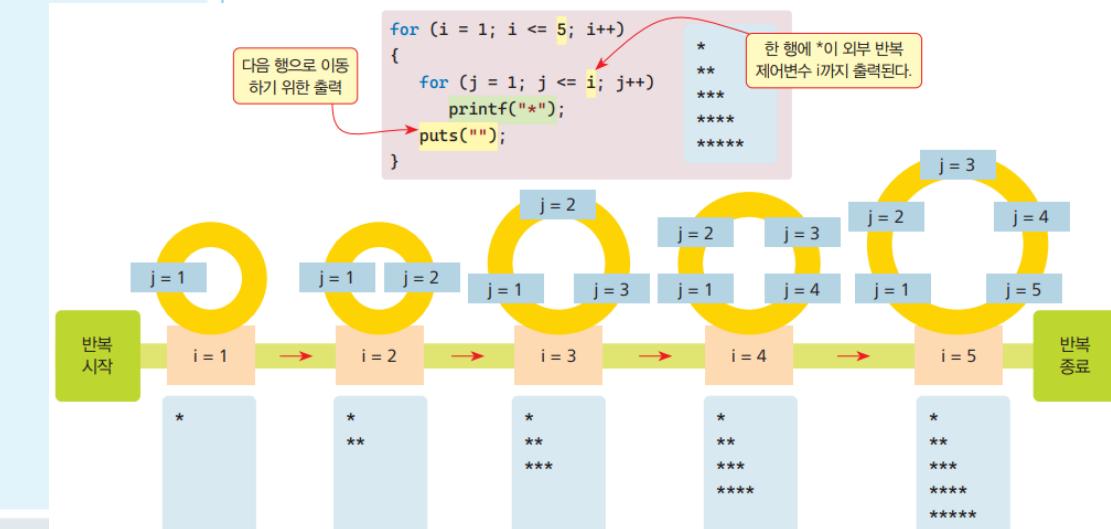
```

01 #include <stdio.h>
02
03 int main(void)
04 {
05     const int MAX = 5;
06     int i, j;
07
08     for (i = 1; i <= MAX; i++)
09     {
10         for (j = 1; j <= i; j++)
11             printf("*");
12         puts("");
13     }
14
15     return 0;
16 }
```

결과

```

*
**
***
****
*****
```



배열

배열의 필요성과 정의

- 배열(array)
 - 여러 변수들이 같은 배열이름으로 일정한 크기의 연속된 메모리에 저장되는 구조
 - 배열을 이용하면 변수를 일일이 선언하는 번거로움이 없어지고
 - 배열을 구성하는 각각의 변수를 반복 구문으로 쉽게 참조 가능
- 배열 정의와 선언 구문
 - 자료유형의 저장공간인 원소를 동일한 크기로 지정된 배열크기만큼 확보한 연속된 저장공간
 - 배열의 중요 요소
 - 배열이름, 원소 자료유형, 배열크기

배열 선언

원소자료형 배열이름[배열크기];

배열크기는 리터럴 상수, 매크로 상수 또는 이들의 연산식이 허용되나 변수는 사용할 수 없다.

```
#define SIZE 5  
  
int score[10];  
double point[20];  
char ch[80];  
float grade[SIZE];  
int score[SIZE+1];  
int degree[SIZE*2];
```

매크로 상수는 결국 리터럴 상수로 바뀌어 컴파일되므로 문제 없이 선언이 가능하다.

배열원소 접근

- 첨자(index)를 이용
 - 배열이름 뒤에 대괄호 사이
 - 첫 번째 배열원소를 참조하는 첨자 값은 0, 다음 두 번째 원소는 1
 - 유효한 첨자의 범위: 0부터 (배열크기-1)까지
 - 배열 선언 시 대괄호 안의 수는 배열크기
 - 선언 이후 대괄호 안의 수는 원소를 참조하는 번호 첨자

```
int score[5];

//배열 원소에 값 저장
score[0] = 78;
score[1] = 97;
score[2] = 85;
//배열 4번째 원소에 값 저장하지 않아 쓰레기값 저장
score[4] = 91;
score[5] = 50; //문법오류는 발생하지 않으나 실행오류 발생
✖ C4789 버퍼 'score'(크기: 20바이트)이(가) 오버런됩니다. 4바이트가 오프셋 20부터 쓰입니다.
```



그림 8-3 배열 선언과 원소 참조

배열원소 일괄 출력

- 배열 선언 후 배열원소에 값을 저장하고 순차적으로 출력

실습예제 8-1 Prj01 01decarray.c 배열 선언 후 배열원소에 값을 저장하고 순차적으로 출력 난이도: ★

```
01 #include <stdio.h>
02
03 #define SIZE 5
04
05 int main(void)
06 {
07     //배열 선언
08     int score[SIZE]; //int score[5];
09
10     //배열 원소에 값 저장
11     score[0] = 78; //첨자를 사용해 배열원소에 저장
12     score[1] = 97;
13     score[2] = 85;
14     //배열 4번째 원소에 값을 대입하지 않아 쓰레기 값 저장
15     score[4] = 91;
16     //score[5] = 50; //문법오류 발생
17
18     //배열원소 출력
19     for (int i = 0; i < SIZE; i++)
20         printf("%d ", score[i]);
21     printf("\n");
22
23     return 0;
24 }
```

결과 78 97 85 -858993460 91

SIZE는 리터럴 상수 또는 매크로 상수로 양의 정수여야 함

첨자가 0에서 4를 벗어나면 문법오류가 발생

초기값을 저장하지 않아 쓰레기 값이 출력됨

배열 초기화

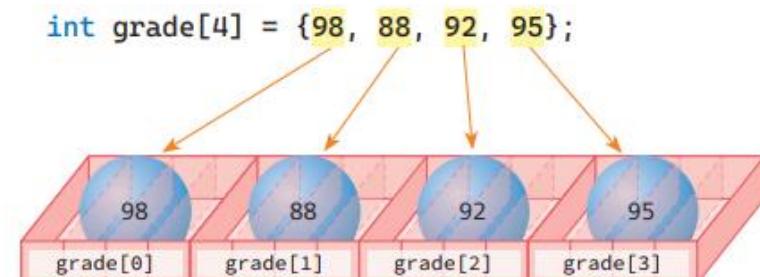
- 배열 선언 초기화

- 배열을 선언하면서 동시에 원소 값을 손쉽게 저장하는 (initialization) 방법
- 중괄호 사이에 여러 원소 값을 쉼표로 구분하여 기술하는 방법
 - 중괄호 사이에는 명시된 배열크기를 넘지 않게 원소 값 나열 가능
- 배열크기는 생략 가능
 - 자동으로 중괄호 사이에 기술된 원소 수가 배열크기
- 원소 값을 나열하기 위해 콤마(,)를 사용하고 전체를 중괄호 {...}로 묶음

원소자료형 배열이름[**배열크기**] = {원소값1, 원소값2, 원소값3, 원소값4, 원소값5, ... } ;

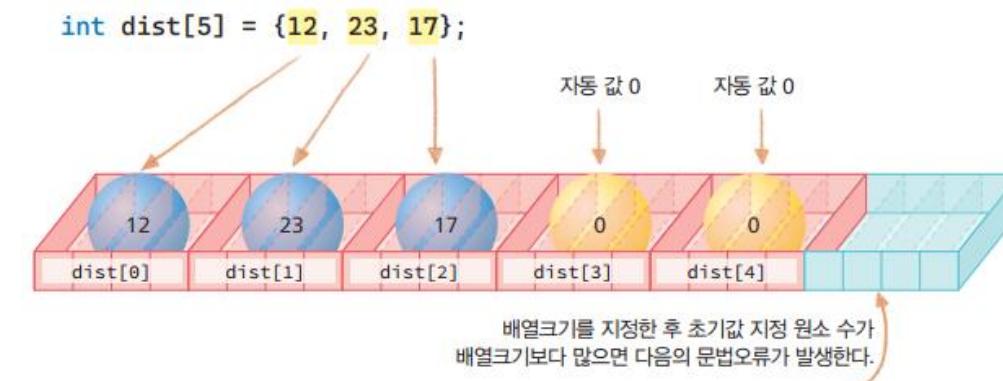
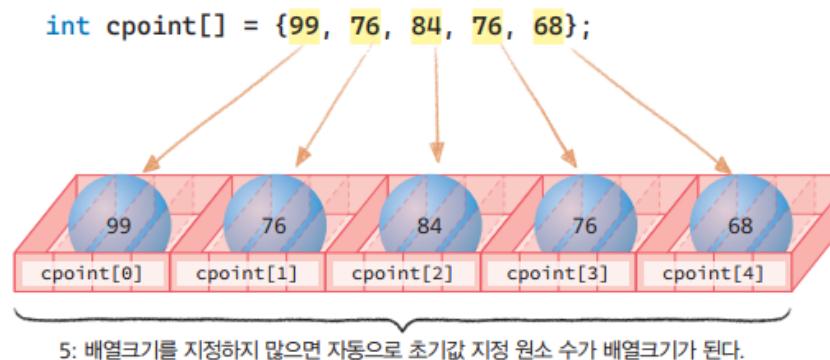
배열크기는 생략 가능하며, 생략 시 원소값의 수가 배열크기가 된다.

```
int grade[4] = {98, 88, 92, 95};
double output[] = {78.4, 90.2, 32.3, 44.6, 59.7, 98.9};
int cpoint[] = {99, 76, 84, 76, 68};
```



배열 기본 값

- 배열크기가 초기값 원소 수보다 크면
 - 지정하지 않은 원소의 초기값은 자동으로 모두 기본값이 저장
 - 기본값이란 자료형에 맞는 0
 - 정수형은 0, 실수형은 0.0 그리고 문자 형은 '\0'인 널문자(문자 코드 변화가 0인 문자)



```
int dist[5] = {12, 23, 17, 55, 57, 71};  
int dist[5] = {0};
```

error C2078: 이니셜라이저가 너무 많습니다.

지정한 배열크기보다 초기값 수가 적으면 모두 0으로 채워지므로 모든 배열 원소가 0으로 채워진다.

배열 선언 초기화를 이용한 합과 평균 출력

실습예제 8-2 Prj02 02initarray.c 배열 선언 초기화를 이용한 합과 평균 출력 난이도: ★

```
01 #include <stdio.h>
02 #define SIZE 6
03 int main(void)
04 {
05     // 배열 score의 선언과 초기화
06     double score[] = { 89.3, 79.2, 84.83, 76.8, 92.52, 97.4 };
07     double sum = 0;
08
09     // for 문을 이용하여 합을 구함
10    for (int i = 0; i < SIZE; i++)
11    {
12        sum += score[i];    // 제어문자 i의 첨자는 0에서 5까지 반복
13        printf("score[%d] = %.2f\n", i, score[i]);
14    }
15    printf("성적의 합은 %.2f이고 평균은 %.2f이다.\n", sum, sum/SIZE);
16
17    return 0;
18 }
```

결과

```
score[0] = 89.30
score[1] = 79.20
score[2] = 84.83
score[3] = 76.80
score[4] = 92.52
score[5] = 97.40
성적의 합은 520.05이고 평균은 86.67이다.
```

배열 선언과 초기화는 두 문장을 나누어 할 수 없으므로, 다음은 컴파일 오류 발생

```
double score[6];
score = { 89.3, 79.2, 84.83, 76.8, 92.52, 97.4 };
```

SIZE는 배열크기인 매크로 상수로 양의 정수인 6으로 정의

다양한 배열 선언 초기화 구문

- 바른 문장과 잘못된 초기화 문장

```
#define SIZE 3
int grade[4] = {98, 88, 92, 95};
double output[SIZE] = {8.4, 0.2, 2.3, 44.6};
int cpoint[] = {99, 76, 84, 76, 68, 93};
char ch[] = {'a', 'b', 'c'};
double width[4] = {23.5, 32.1};
```

올바른 초기화 문장



TIP 배열 선언 초기화에서 주의점

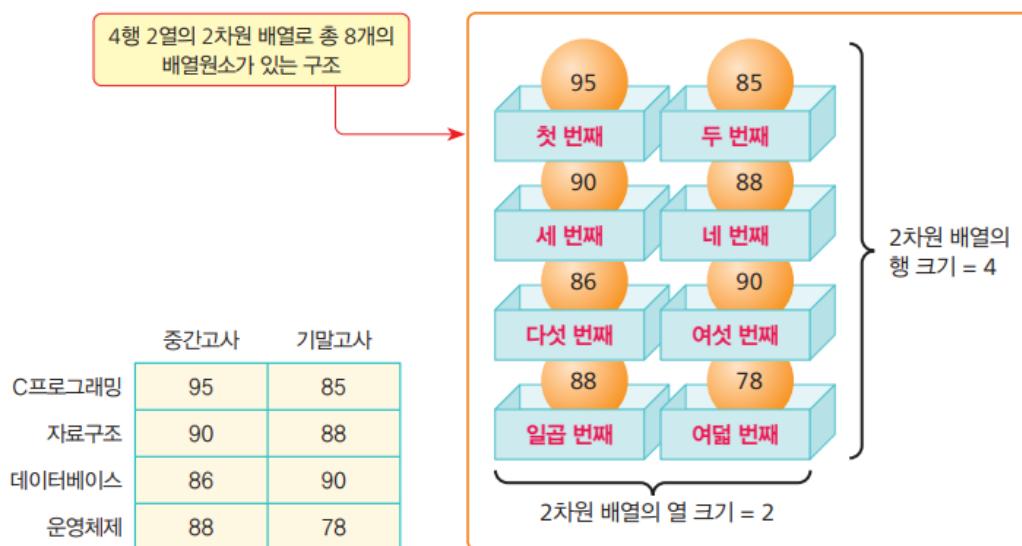
다음은 오류가 발생하는 배열 선언 초기화 문장이다. 초기화에서도 변수와 const 상수는 배열크기로 사용할 수 없다. 마지막 문장은 초보자가 자주 실수하는 문장으로, 중괄호를 사용한 초기화 방법은 반드시 배열 선언 시에만 이용이 가능하며 배열 선언 이후에는 사용할 수 없으니 주의하자.

표 8-2 배열 선언 초기화 시 오류 발생과 원인

변수와 배열 선언 문장	설명 및 오류 원인
int n = 5; const int size = 6;	변수 n과 const 상수 size 선언
int score[n] = {89, 92, 91}; int cpoint[size] = {3, 5, 7};	변수 n은 배열크기로 사용 불가 상수 변수 size는 배열크기로 사용 불가
int grade[3] = {98, 88, 92, 95};	원소 수 4가 배열크기 3보다 큼
int cpoint[] = {99 76 84 76 68 93};	원소값을 구분하는 콤마(,)가 빠짐
char ch[] = {a, b, c};	원소값인 a, b, c가 문자여야 함
double width[4]; width = {23.5, 32.1};	배열 선언 이후에는 중괄호를 사용한 초기화를 사용할 수 없으며, 배열 선언 시 double width[4] = {23.5, 32.1};로는 가능

2차원 배열 개요

- 2차원 배열은 테이블 형태의 구조
 - 행(row)과 열(column)의 구조로 표현



2차원 배열 선언

원소자료형 배열이름[배열행크기][배열열크기];

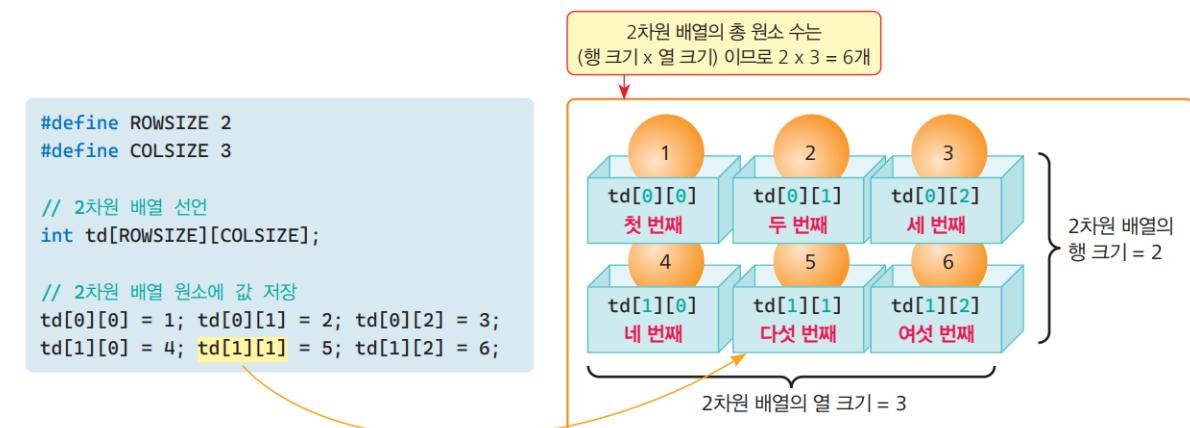
배열 선언 시 배열크기는 생략할 수 없으며
배열크기는 리터럴 상수, 매크로 상수
또는 그들의 연산식이 허용된다.

```
#define RSIZE 5
#define CSIZE 2

int score[RSIZE][CSIZE];
double point[2][3];
char ch[5][80];
float grade[7][CSIZE];
```

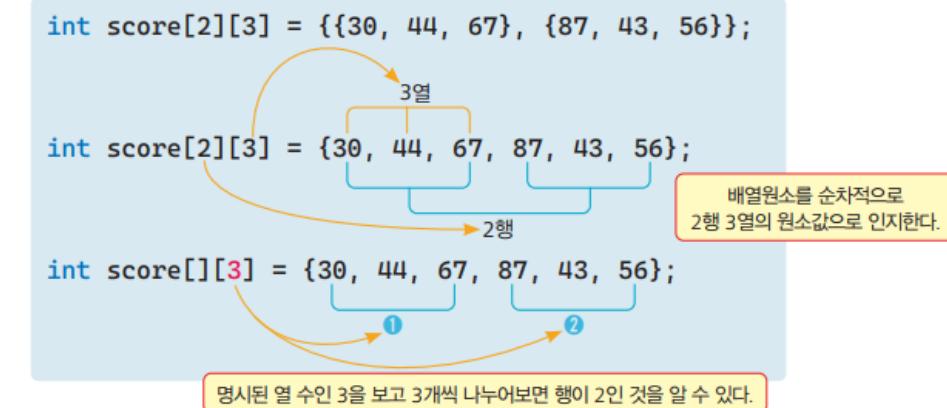
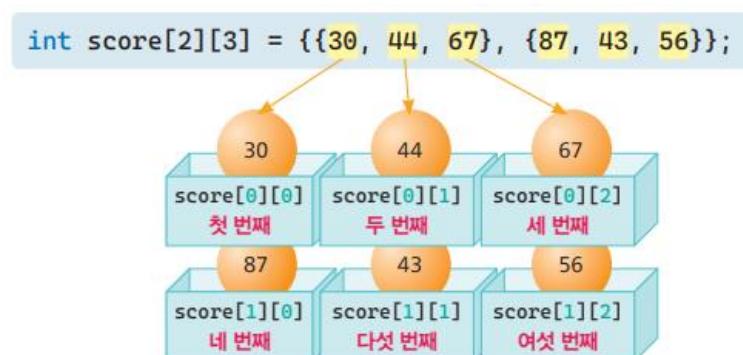
2차원 배열 구조

- 배열 선언 `int td[2][3];`
 - `td[0][0]`으로 첫 번째 원소를 참조
 - 두 번째 원소는 `td[0][1]`,
 - 두 번째 행의 첫 항목인 네 번째 원소는 `td[1][0]`으로 행 첨자가 1 증가
 - 행 첨자는 0에서 (`행 크기-1`)까지 유효
 - 마찬가지로 열 첨자는 0에서 (`열크기 - 1`)까지 유효
- 행 우선 (row major) 배열
 - 행을 먼저 배치하는 특징
 - 첫 번째 행의 모든 원소가 메모리에 할당된 이후
 - 두 번째 행의 원소가 순차적으로 할당



2차원 배열 선언 초기화

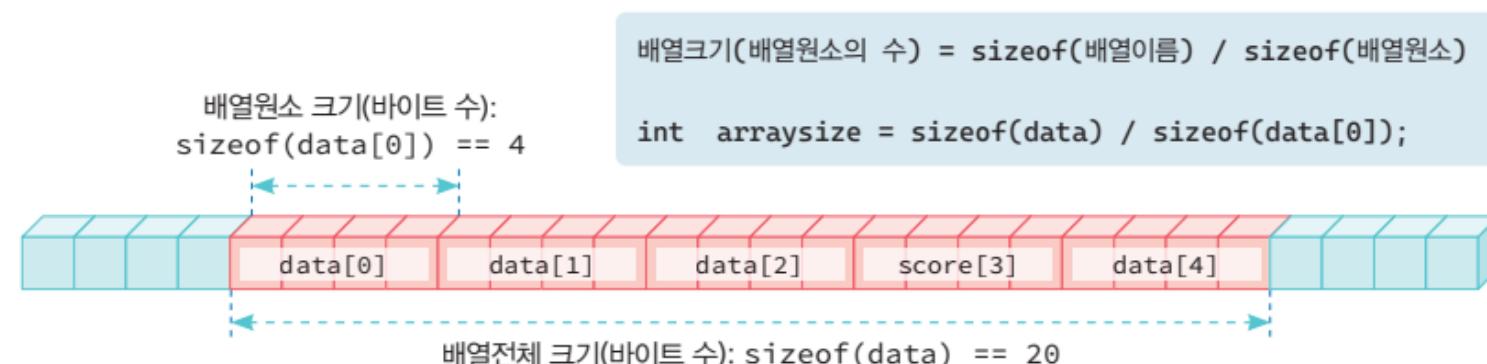
- 첫 번째 방법은 중괄호를 중첩되게 이용
 - 중괄호 내부에 행에 속하는 값을 다시 중괄호로 묶고, 중괄호와 중괄호 사이에는 쉼표로 분리
 - 행인 중괄호 내부의 초기값들은 쉼표로 분리
 - 2차원 구조를 행과 열로 표현 할 수 있는 장점
- 다른 방법
 - 1차원 배열과 같이 하나의 중괄호로 모든 초기 값을 쉼표로 분리하는 방법



배열크기 연산

- 배열크기 계산방법
 - 연산자 `sizeof`를 이용한 식 (`sizeof(배열이름)` / `sizeof(배열원소)`)
 - 저장공간의 크기를 바이트 수로 반환하는 연산자 `sizeof`
 - `sizeof(배열이름)`
 - 배열의 전체 공간의 바이트 수
 - `sizeof(배열원소)`
 - 배열원소 하나의 바이트 수

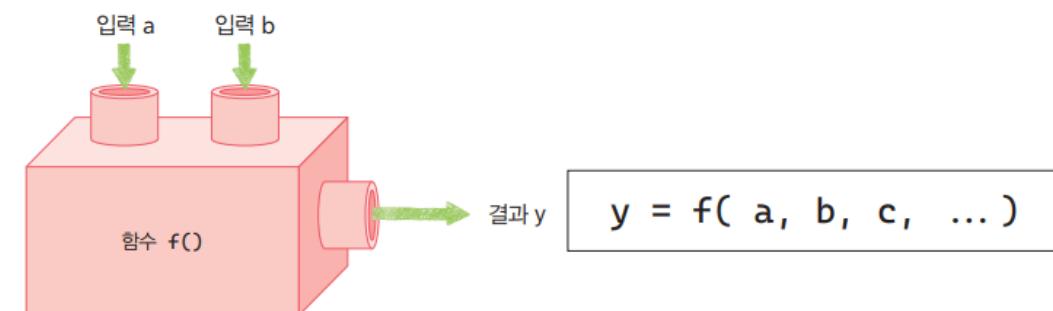
```
int data[] = {12, 23, 17, 32, 55};
```



함수

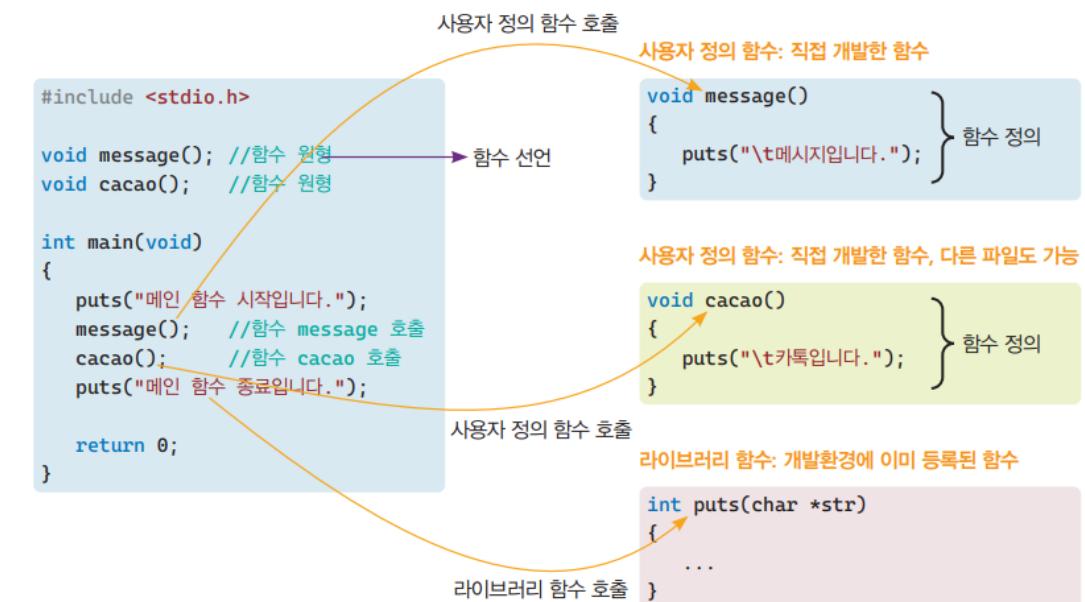
함수의 이해

- **함수 개념**
 - **함수(function)**
 - 프로그램에서 원하는 특정한 작업을 수행하도록 설계된 독립된 프로그램 단위
 - 필요한 입력을 받아 원하는 어떤 기능을 수행한 후 결과를 반환(return)하는 프로그램 단위
 - **라이브러리 함수(library function)와 사용자 정의 함수(user defined function)로 구분**
 - **사용자 정의 함수**
 - 필요에 의해서 개발자가 직접 개발하는 함수
- **C 프로그램**
 - 여러 함수로 구성되는 프로그램
 - 최소한 main() 함수와 필요한 다른 함수로 구성되는 프로그램
 - **함수 main()**
 - 이름이 지정된 함수로서 프로그램의 실행이 시작되는 특수한 함수



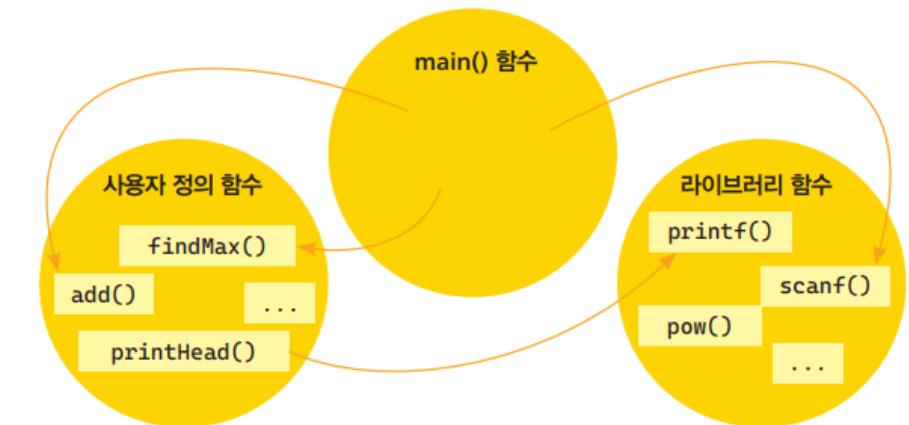
함수 정의와 호출

- C 프로그램 main() 함수
 - 첫 줄에서 시작하여 마지막 줄을 마지막으로 실행한 후 종료
 - 사용자가 직접 개발한 함수를 사용
 - 함수 선언(function declaration)과 함수 호출(function call), 함수 정의(function definition)가 필요
- 하나의 응용 프로그램
 - 하나의 main() 함수와 여러 개의 다른 함수로 구성
 - 필요에 따라 여러 소스 파일로 나누어 프로그래밍 가능



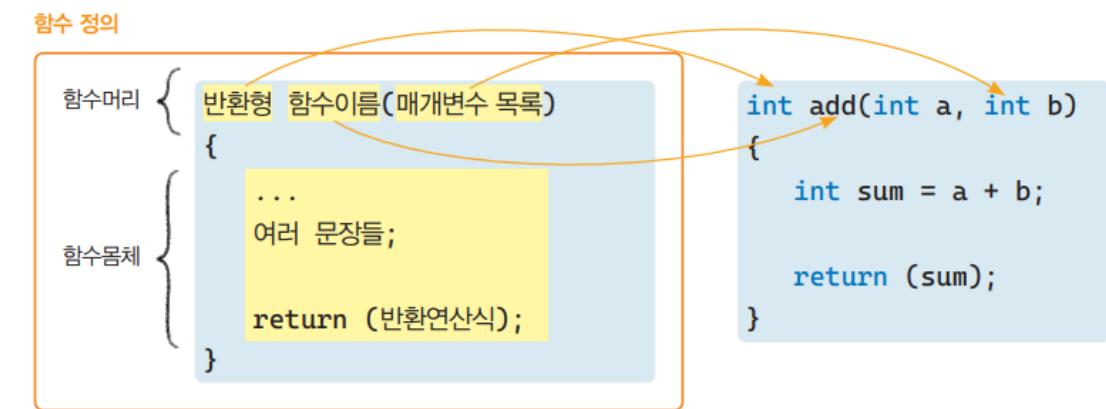
절차적 프로그래밍

- 주어진 문제를 여러 개의 작은 문제로 나누어 해결 하듯
 - 하나의 프로그램은 여러 함수로 나누어 프로그래밍
 - 문제 분해(problem decomposition)
 - 하나의 프로그램을 작은 단위의 여러 함수로 나누는 것
 - 문제 해결의 한 방법
- 절차적 프로그래밍(procedural programming) 방식
 - 함수 중심의 프로그래밍 방식
 - 모듈화 프로그램 (modular program) 또는 구조화된 프로그램(structured program)
 - 적절한 함수로 잘 구성된 프로그램



함수 정의 구문

- 함수머리(function header)와 함수몸체(function body)로 구성
- 함수머리(function header)
 - 반환형과 함수이름, 매개변수 목록으로 구성
 - 반환형: 함수 결과값의 자료형
 - 간단히 반환형
 - 함수이름: 식별자의 생성규칙
 - 매개변수 목록: ‘자료형 변수이름’의 쌍
 - 필요한 수만큼 콤마로 구분하여 기술
- 함수몸체(function body)
 - {...}와 같이 중괄호로 시작하여 중괄호로 종료
 - 함수가 수행해야 할 문장들로 구성
 - 마지막은 대부분 결과값을 반환하는 return 문장으로 종료
 - 결과값이 없다면 생략 가능



반환형과 return

- 함수에서 반환값을 전달하는 목적과 함께 함수의 작업 종료를 알리는 문장
 - 함수가 반환값이 없다면 반환형으로 void를 기술

```
int findMin(int x, int y)
{
    int min = x < y ? x : y;

    return (min);
}

void printMin(int a, int b)
{
    int min = a < b ? a : b;
    printf("%n", min);

    return;      //생략 가능
}
```

return 문장

return (반환연산식);

```
return 0;
return (a + b);
return 2 * PI * r;
return ;
```

함수원형

- function prototype
 - 함수도 정의된 함수를 호출하기 이전에 필요
 - 함수원형은 함수를 선언하는 문장
- 구문
 - 함수머리에 세미콜론을 넣은 문장
 - int add(int a, int b);
 - int add(int, int);

함수원형

반환형 함수이름(매개변수 목록);

함수원형은 문장이므로 마지막에
세미콜론 ;이 반드시 필요하다.

```
int add(int a, int b);
int add(int, int);
int findMin(int x, int y);
int findMin(int, int);
```

함수원형을 함수 main() 위에 배치

```
#include <stdio.h>
```

함수 선언

```
int add(int a, int b);
//int add(int, int)도 가능
```

```
int main(void)
```

```
{
```

```
    int a = 3, b = 5;
```

```
    int sum = add(a, b);
```

함수 호출

```
    ...
```

```
    return 0;
```

```
}
```

```
int add(int a, int b)
```

```
{
```

```
    int sum = a + b;
```

```
    return (sum);
```

```
}
```

함수원형을 함수 main() 내부에 배치

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

함수 선언

```
int add(int a, int b);
//int add(int, int)도 가능
```

```
int a = 3, b = 5;
```

```
int sum = add(a, b);
```

함수 호출

```
    ...
```

```
    return 0;
```

```
}
```

```
int add(int a, int b)
```

```
{
```

```
    int sum = a + b;
```

```
    return (sum);
```

```
}
```

함수의 정의와 호출

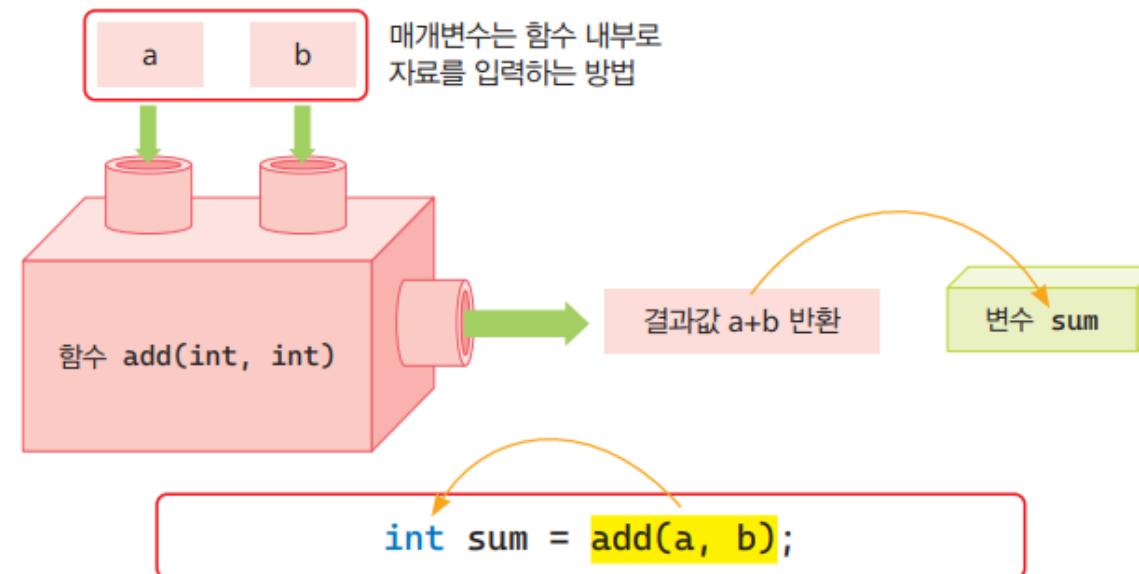
실습예제 9-2 Prj02 02funcadd.c 함수의 정의와 호출 난이도: ★

```
01 #include <stdio.h>
02
03 //int add(int a, int b); //이 위치도 가능
04
05 int main(void)
06 {
07     int a = 3, b = 5;
08     int add(int a, int b); //int add(int, int)도 가능
09
10    //위 함수원형이 없으면 함수 호출에서 경고 발생
11    int sum = add(a, b);
12    printf("합: %d\n", sum);
13
14    return 0;
15 }
16
17 //함수 add의 함수 구현 또는 함수 정의 부분
18 int add(int a, int b)
19 {
20     int sum = a + b; //변수 sum의 값을 반환하는 return 문장으로
21                         //괄호는 생략 가능하며, 함수 호출한 곳으로
22                         //매개변수는 a+b의 결과를 반환
23     return (sum); //괄호는 생략 가능
24 }
25
26 //위 main() 함수에서 호출이 없으므로 이 함수 구현은 실행되지 않음
27 int findMin(int x, int y)
28 {
29     int min = x < y ? x : y;
30
31     return (min);
32 }
```

결과 합: 8

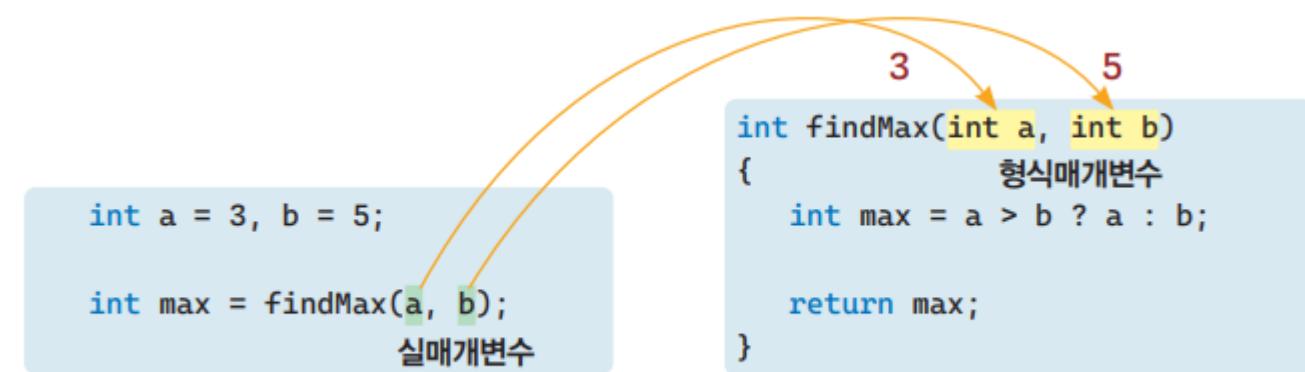
매개변수 정의

- **함수 매개변수(parameter)**
 - 함수를 호출하는 부분에서 함수몸체로 값을 전달할 목적으로 이용
 - 자료형과 변수명의 목록으로 표시
 - 필요 없으면 키워드 void를 기술
- **반환값**
 - 함수를 호출하는 부분에서 함수가 작업을 수행한 후, 다시 함수를 호출한 영역으로 보내는 결과 값



형식매개변수와 실매개변수

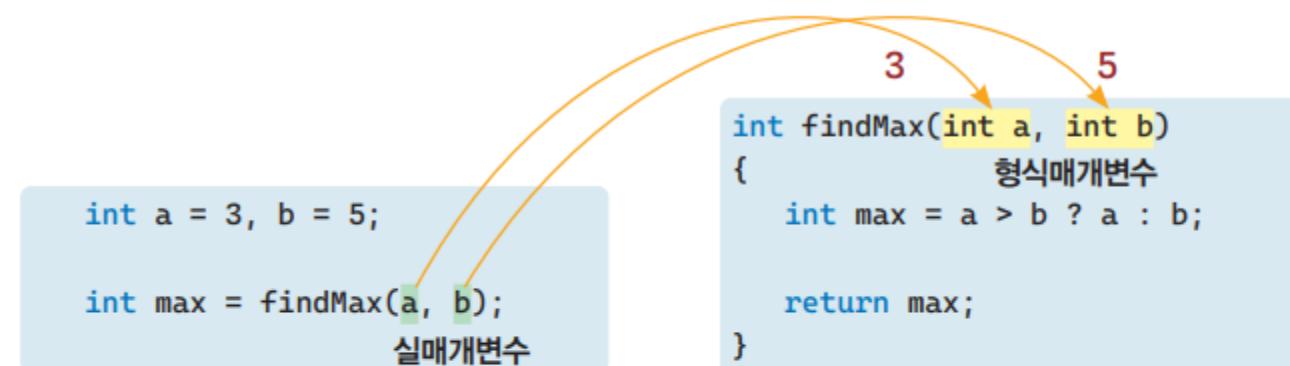
- 형식매개변수(formal parameters)
 - 함수 정의에서 기술되는 매개변수 목록의 변수
 - 함수 내부에서만 사용될 수 있는 변수
- 실매개변수(real parameters)와 실인자(real argument)
 - 함수를 호출할 때 기술되는 변수 또는 값
 - 간단히 인자(argument)라고도 부름
- 일반적으로 매개변수, 인자, 인수
 - 구분하지 않고 사용하는 경우도 많음



함수 호출 시 매개변수의 수와 자료형이 다르면 문법오류가 발생한다.

값에 의한 호출(call by value)

- 함수가 호출되는 경우 실인자의 값이 형식인자의 변수에 각각 복사된 후 함수가 실행
 - 형식인자 a, b와는 전혀 다른 변수
 - 함수에서 매개변수(일반 변수)의 값을 수정하더라도 함수를 호출한 곳에서의 실제 변수의 값은 변화되지 않는 특징



함수 호출 시 매개변수의 수와 자료형이 다르면 문법오류가 발생한다.

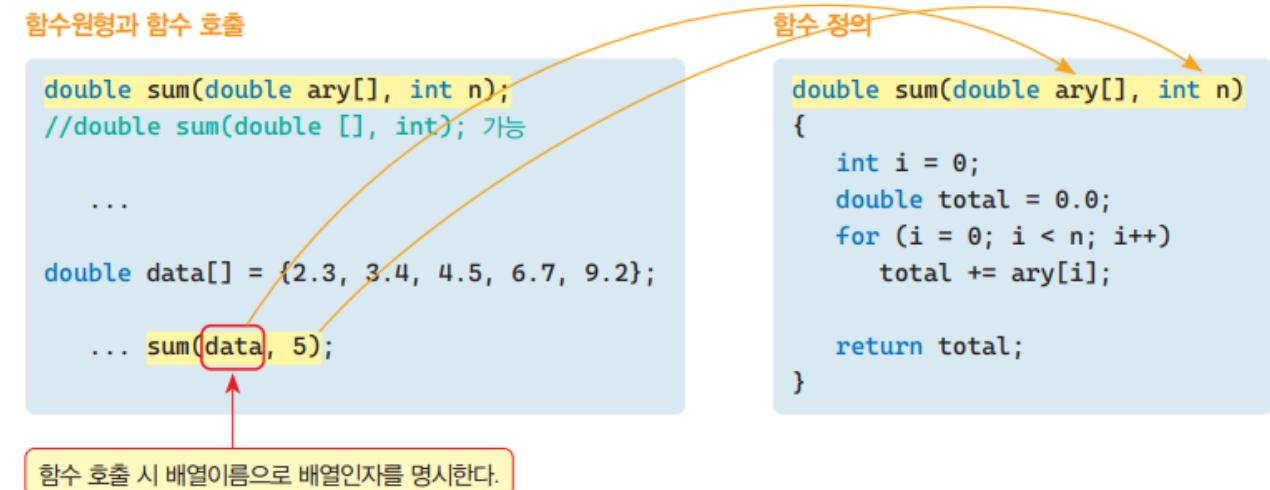
```
int max = findMax();          //오류 발생
int max = findMax(2);         //오류 발생
int max = findMax(2.4);       //오류 발생
int max = findMax(2.4, 6.8);  //오류 발생
```

함수의 정의와 호출

실습예제 9-3	Prj03	03functioncall.c 03others.c	함수의 정의와 호출	난이도: ★
		소스파일	03functioncall.c	
			<pre> 01 #include <stdio.h> 02 03 int add(int a, int b); //int add(int, int)도 가능 04 int findMax(int, int); //int findMax(int a, int b)도 가능 05 void printMin(int, int); //int printMin(int a, int b)도 가능 06 07 int main(void) 08 { 09 int a = 10, b = 15; 10 11 int max = findMax(a, b); 12 printf("최대: %d\n", max); 13 printf("합: %d\n", add(a, b)); 14 15 //반환값이 없는 함수 호출은 일반문장처럼 사용 16 printMin(a, b);← 17 18 return 0; 19 } 20 21 void printMin(int a, int b) 22 { 23 int min = a < b ? a : b; 24 printf("최소: %d\n", min); 25 }</pre>	
		소스파일	03others.c	
			<pre> 01 //함수 add, findMax, findMin, printMin 구현 02 int add(int a, int b) 03 { 04 int sum = a + b; 05 06 return (sum); 07 } 08 09 int findMax(int a, int b) 10 { 11 int max = a > b ? a : b; 12 13 return max; 14 } 15 16 int findMin(int x, int y) 17 { 18 int min = x < y ? x : y; 19 20 return (min); 21 }</pre>	
				결과
				최대: 15 합: 25 최소: 10

매개변수로 배열 사용

- 함수의 매개변수로 배열을 전달
 - 한 번에 여러 개의 변수를 전달하는 효과
- 함수 sum()
 - 실수형 배열의 모든 원소의 합을 구하여 반환하는 함수
 - 형식매개변수는 실수형 배열 double ary[]와 배열크기 int n
 - 첫 번째 형식매개변수에서 배열 자체에 배열크기를 기술하는 것은 아무 의미가 없음
 - double ary[5]보다는 double ary[]라고 기술하는 것을 권장
 - 실제로 함수 내부에서 실인자로 전달되는 배열크기를 알 수 없으므로
 - 배열크기를 두 번째 인자로 사용



배열을 매개변수로 사용하는 방법

- 함수 크기를 제외한 int ary[]로 기술
 - 배열의 크기도 매개변수로 함께 전달
 - 함수 이름인 point 또는 &point[0]를 사용

```
printf("함수 sumary() 호출로 구한 합은 %d\n", sumary(point, aryLength));
printf("함수 sumary() 호출로 구한 합은 %d\n", sumary(&point[0], aryLength));
```

배열의 크기는 의미가 없어 어떤 수를 써도 상관없으므로 생략하고 배열크기를 다른 인자로 사용

```
int sumary(int ary[], int SIZE)
{
    ...
}
```

```
int sumary(int ary[10], int SIZE)
{
    ...
}
```

```
for (i = 0; i < SIZE; i++)
{
    sum += ary[i];
}
```

함수에서 배열인자를 사용

실습예제 9-4 Prj04 04aryparam.c 함수에서 배열인자를 사용 난이도: ★

```
01 #include <stdio.h>
02
03 int sumary(int ary[], int SIZE);
04
05 int main(void)
06 {
07     int point[] = { 95, 88, 76, 54, 85, 33, 65, 78, 99, 82 };
08
09     int arySize = sizeof(point);
10    printf("메인에서 배열 전체크기: %d\n", arySize);
11    int aryLength = arySize / sizeof(int);
12
13    int sum = 0;
14    for (int i = 0; i < aryLength; i++)
15        sum += point[i];
16
17    printf("메인에서 구한 합은 %d\n", sum);
18    printf("함수 sumary() 호출로 구한 합은 %d\n", sumary(point, aryLength));
19    printf("함수 sumary() 호출로 구한 합은 %d\n", sumary(&point[0], aryLength));
20
21    return 0;
22 }
23
24 int sumary(int ary[], int SIZE)
25 {
26     int sum = 0;
27     printf("함수에서 배열 전체크기: %d\n", (int) sizeof(ary));
28     for (int i = 0; i < SIZE; i++)
29     {
30         sum += ary[i];
31     }
32
33     return sum;
34 }
```

결과

```
메인에서 배열 전체크기: 40
메인에서 구한 합은 755
함수에서 배열 전체크기: 8
함수 sumary() 호출로 구한 합은 755
함수에서 배열 전체크기: 8
함수 sumary() 호출로 구한 합은 755
```

배열의 실제 개변수 사용은 point와 &point[0]으로 가능

함수의 매개변수에 배열을 사용하더라도 배열의 첫 원소 주소만이 전달되므로 배열의 크기를 전달해야 배열에 관한 처리가 가능함. 이것은 배열 크기를 구하는 문장으로, 변수 aryLength에는 배열 크기가 저장됨

이 곳에서는 함수를 호출한 곳에서의 배열 전체 크기를 알 수 없음

이차원 배열을 함수 인자로 이용하는 방법

- 함수원형과 함수정의의 헤더
 - 첫 번째 대괄호 내부의 크기를 제외한 다른 모든 크기는 반드시 기술

함수원형과 함수 호출

```
...
//2차원 배열값을 모두 더하는 함수원형
double sum(double data[][3], int, int);
//2차원 배열값을 모두 출력하는 함수원형
void printarray(double data[][3], int, int);

...
double x[][3] = { {11, 12}, {21, 22},
{31, 32} };

int rowsize = sizeof(x) / sizeof(x[0]);
int colszie = sizeof(x[0]) / sizeof(x[0][0]);

printarray(x, rowsize, colszie);
... sum(x, rowsize, colszie) ...
...
```

함수 정의

```
//2차원 배열값을 모두 출력하는 함수
void printarray(double data[][3], int rowsize, int colszie)
{
}

//2차원 배열값을 모두 더하는 함수
double sum(double data[][3], int rowsize, int colszie)
{
    ...
    for (i = 0; i < rowsize; i++)
        for (j = 0; j < colszie; j++)
            total += data[i][j];
    return total;
}
```

배열의 전체 원소 수: 12
 $\text{sizeof}(x) / \text{sizeof}(x[0][0])$

배열의 행 수: 4
 $\text{sizeof}(x) / \text{sizeof}(x[0])$

배열의 열 수: 3
 $\text{sizeof}(x[0]) / \text{sizeof}(x[0][0])$

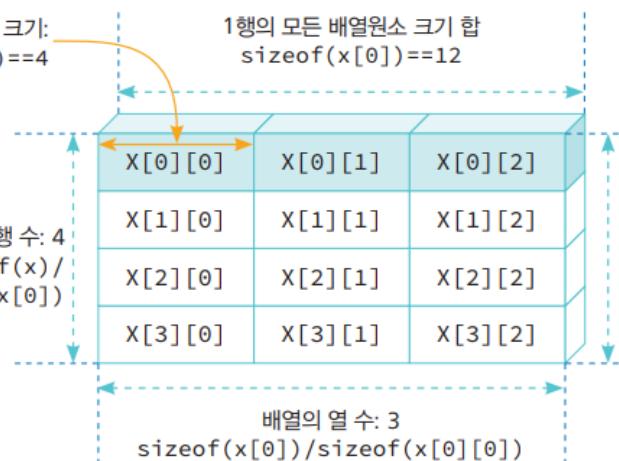
배열의 한 원소의 크기:
 $\text{sizeof}(x[0][0]) == 4$

1행의 모든 배열원소 크기 합
 $\text{sizeof}(x[0]) == 12$

배열의 행 수: 4
 $\text{sizeof}(x) / \text{sizeof}(x[0])$

배열의 열 수: 3
 $\text{sizeof}(x[0]) / \text{sizeof}(x[0][0])$

배열 전체 크기:
 $\text{sizeof}(x)$



2차원 배열을 인자로 하는 함수의 정의와 호출

실습예제 9-5 Prj05 05twodarrayfunc.c 2차원 배열을 인자로 하는 함수의 정의와 호출 난이도: ★★

```

01 #include <stdio.h>
02
03 double sum(double data[][2], int, int); //2차원 배열값을 모두 더하는 함수원형
04 void printarray(double data[][2], int, int); //2차원 배열값을 모두 출력하는 함수원형
05
06 int main(void)
07 {
08     //3 x 2 행렬을 위한 2차원 배열 선언 및 초기화
09     double x[][2] = { { 11, 12 }, { 21, 22 }, { 31, 32 } };
10
11     int rowsize = sizeof(x) / sizeof(x[0]);
12     int colszie = sizeof(x[0]) / sizeof(x[0][0]);
13     printf("2차원 배열의 자료값은 다음과 같습니다.\n");
14     printarray(x, rowsize, colszie);
15     printf("2차원 배열 원소합은 %.3lf 입니다.\n", sum(x, rowsize, colszie));
16
17     return 0;
18 }
19
20 //배열값을 모두 더하는 함수
21 double sum(double data[][2], int rowsize, int colszie)
22 {
23     double total = 0;
24     for (int i = 0; i < rowsize; i++)
25         for (int j = 0; j < colszie; j++)
26             total += data[i][j];
27
28     return total;
29 }
30
31 //배열값을 모두 출력하는 함수
32 void printarray(double data[][2], int rowsize, int colszie)
33 {
34     for (int i = 0; i < rowsize; i++)
35     {
36         printf("%d행원소: ", i + 1);
37         for (int j = 0; j < colszie; j++)
38             printf("x[%d][%d] = %5.2lf ", i, j, data[i][j]);
39         printf("\n");
40     }
41     printf("\n");
42 }
```

결과

2차원 배열의 자료값은 다음과 같습니다.

1행원소: x[0][0] = 11.00 x[0][1] = 12.00

2행원소: x[1][0] = 21.00 x[1][1] = 22.00

3행원소: x[2][0] = 31.00 x[2][1] = 32.00

2차원 배열 원소합은 129.000 입니다.

재귀 함수와 재귀적 특성

- 재귀 함수(recursive function)
 - 함수구현에서 자신의 함수를 호출하는 함수
- 재귀적 특성을 표현하는 알고리즘
 - 재귀 함수를 이용하면 문제를 쉽게 해결
 - n의 계승(n factorial)을 나타내는 수식 $n!$ 은 $1 * 2 * 3 * \dots * (n-2) * (n-1) * n$ 을 의미하는 수식이다.
즉 $n!$ 의 정의에서 보듯 계승은 $n!$ 을 정의하는 데 $(n-1)!$ 을 사용하는 재귀적 특성을 갖는다.

$$n! \begin{cases} 0! = 1 \\ n! = n * (n-1)! \text{ for } (n \geq 1) \end{cases}$$

```
if (n <= 1)
    n! = 1
else
    n! = n * (n-1)!
```



```
int factorial(int num)
{
    if (num <= 1)
        return 1;
    else
        return (num * factorial(num - 1));
}
```

n!을 구현한 재귀함수

- 재귀함수 factorial()을 이용하여 1!에서 10!까지 결과를 출력하는 예제

실습예제 9-6 Prj06 06factorial.c n!을 구현한 재귀함수 난이도: ★

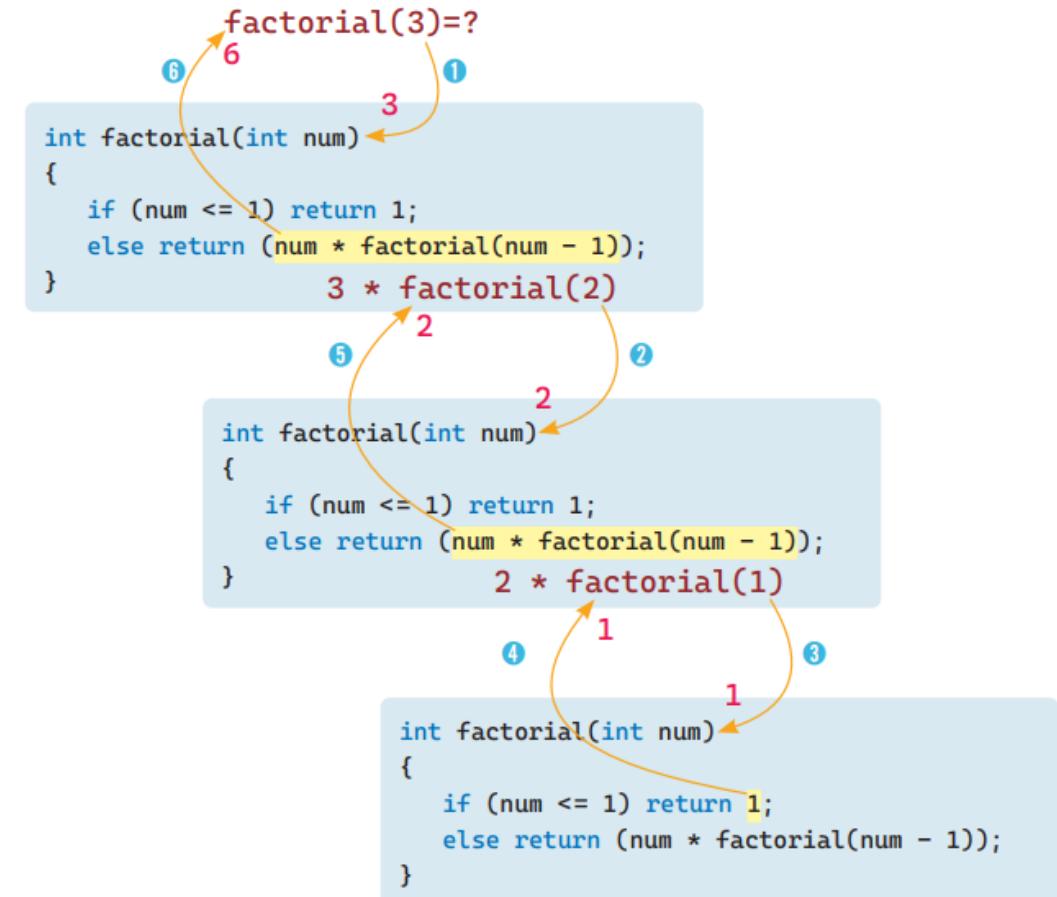
```
01 #include <stdio.h>
02
03 int factorial(int); //함수원형
04
05 int main(void)
06 {
07     for (int i = 1; i <= 10; i++)
08         printf("%2d! = %d\n", i, factorial(i));
09
10     return 0;
11 }
12
13 // n! 구하는 재귀함수
14 int factorial(int number)
15 {
16     if (number <= 1) ← 조건식을 만족하면 더 이상 자기 자신인
17         return 1;    함수 factorial()을 호출하지 않는다.
18     else
19         return (number * factorial(number - 1));
20 }
```

결과

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```

함수 factorial(3)을 호출한 경우 실행 과정

- factorial(3)을 호출
 - 함수 factorial(3) 내부에서 다시 factorial(2)를 호출
 - factorial(2)에서는 다시 factorial(1)을 호출
 - 결국 factorial(1) 내부에서 return 1을 실행
 - 다시 factorial(2)로 돌아와 반환 값 1을 이용하여 return (2*1)을 실행
 - 계속해서 factorial(3)으로 돌아와 factorial(2)의 결과값인 2를 이용하여 return (3*2)를 실행하 결국 6을 반환
 - 즉 3!의 결과 6
- 3!을 구하기 위해 내부적으로 2번의 함수 호출



난수와 구현

- 난수(random number)
 - 특정한 나열 순서나 규칙을 가지지 않는 연속적인 임의의 수
 - 임의의 수란 어느 수가 반환될지 예측할 수 없으며
 - 어느 수가 선택될 확률이 모두 동일하다는 의미
 - 주사위나 로또 복권 당첨기가 같은 기구가 난수를 발생시키는 장치
- 난수를 생성하는 함수 rand()를 이용
 - 로또 프로그램 등 임의의 수가 필요로 하는 다양한 프로그래밍에 활용
 - 함수원형은 헤더파일 stdlib.h(standard library)에 정의
 - 0에서 32767사이의 정수 중에서 임의로 하나의 정수를 반환

실습예제 9-7 Prj07 07rand.c 난수를 위한 함수 rand()의 이용 난이도: ★

```

01 #include <stdio.h>
02 #include <stdlib.h> //rand()를 위한 헤더파일 포함
03
04 int main(void)
05 {
06     printf("0 ~ %5d 사이의 난수 8개: rand()\n", RAND_MAX);
07     for (int i = 0; i < 8; i++)
08         printf("%5d ", rand());
09     puts("");
10
11     return 0;
12 }
```

기호 상수 RAND_MAX를
16진수 0x7fff로 정의

결과 0 ~ 32767 사이의 난수 8개: rand()
41 18467 6334 26500 19169 15724 11478 29358

여러 번 실행에도 항상
같은 수가 출력

srand()로 시드 값을 먼저 지정

- 함수 rand()는 함수호출 순서에 따라 항상 일정한 수가 반환
 - 항상 41, 18467, 6334, 26500, 19169 등의 값 들이 출력
- 매번 난수를 다르게 생성하려면
 - 시드(seed) 값을 이용
 - 시드 값이란 난수를 다르게 만들기 위해 처음에 지정하는 수로서 시드 값이 다르면 난수가 달라짐
- srand(seed)를 호출
 - 먼저 서로 다른 시드 값인 seed를 이용
 - 항상 서로 다른 seed 값을 지정하기 위해 현재 시간의 함수 time()을 이용
 - 함수 time(NULL)은 1970년 1월 1일 이후 현재까지 경과된 시간을 초 단위로 반환

실습예제 9-8 Prj08 08srand.c srand()로 시드 값을 먼저 지정한 후 1에서 100 사이의 난수를 생성 나이도: ★

```

01 #include <stdio.h>
02
03 #include <stdlib.h> //rand(), srand()을 위한 헤더파일 포함
04 #include <time.h>   //time()을 위한 헤더파일 포함
05
06 #define MAX 100
07
08 int main(void)
09 {
10     long seconds = (long) time(NULL);
11     srand(seconds);
12
13     printf("1 ~ %5d 사이의 난수 8개:\n", MAX);
14     for (int i = 0; i < 8; i++)
15         printf("%5d ", rand() % MAX + 1);
16     puts("");
17
18     return 0;
19 }
```

결과 1 ~ 100 사이의 난수 8개:
59 47 43 94 40 63 54 4

헤더파일 math.h

- 수학 관련 함수를 사용하려면 헤더파일 math.h을 삽입

함수	처리 작업
double sin(double x)	삼각함수 sin
double cos(double x)	삼각함수 cos
double tan(double x)	삼각함수 tan
double sqrt(double x)	제곱근, square root(x)
double exp(double x)	e^x
double log(double x)	$\log_e(x)$
double log10(double x)	$\log_{10}(x)$
double pow(double x, double y)	x^y
double ceil(double x)	x보다 작지 않은 가장 작은 정수(천정 값)
double floor(double x)	x보다 크지 않은 가장 큰 정수(바닥 값)
int abs(int x)	정수 x의 절대 값
double fabs(double x)	실수 x의 절대 값

실습예제 9-9 Prj09 09math.c 다양한 수학 관련 함수 난이도: ★

```

01 #include <stdio.h>
02 #include <math.h> //수학 관련 다양한 함수머리 포함 헤더파일
03
04 int main(void)
05 {
06     printf(" i i제곱 i세제곱 제곱근(sqrt)\n");
07     printf("-----\n");
08     for (int i = 3; i < 7; i++)
09         printf("%3d %7.1f %9.1f %9.1f\n", i, pow(i, 2), pow(i, 3), sqrt(i));
10     printf("\n");
11
12     printf("%5.2f, ", exp(1.0));
13     printf("%5.2f, ", pow(3.14, 1.0)); 함수 sqrt(81)는  $\sqrt{81}$  반환
14     printf("%5.2f\n", sqrt(81));
15     printf("%5.2f, ", ceil(3.6)); 함수 ceil(3.6)은 3.6의 천정 값인 4.0을 반환하는데,
16     printf("%5.2f, ", floor(5.8)); 천정 값 ceil(x)이란 x보다 작지 않은 가장 작은 정수를 뜻함
17     printf("%5.2f\n, ", fabs(-10.2)); 바닥 값 floor(x)이란 x보다 크지 않은 가장 큰 정수를 뜻함
18
19     return 0;
20 }
```

Microsoft Visual Studio 디버그 콘솔

```

3 9.0 27.0 1.7
4 16.0 64.0 2.0
5 25.0 125.0 2.2
6 36.0 216.0 2.4

2.72, 3.14, 9.00
4.00, 5.00, 10.20
,
```

헤더파일 ctype.h

- 문자 관련 함수는 헤더파일 ctype.h에 매크로로 정의

표 9-2 헤더파일 ctype.h의 주요 문자 관련 함수 매크로

함수원형	기능
isalpha(char)	영문자 검사
isupper(char)	영문 대문자 검사
islower(char)	영문 소문자 검사
isdigit(char)	숫자(0~9) 검사
isxdigit(char)	16진수 숫자(0~9, A-F, a-f) 검사
isspace(char)	공백(' ', '\n', '\t', '\f', '\v', '\r') 문자 검사
ispunct(char)	구두(빈칸이나 알파뉴메릭(alphanumeric)을 제외한 출력문자) 문자 검사
isalnum(char)	영문과 숫자(alphanumeric)(0~9, A~Z, a~z) 검사
isprint(char)	출력 가능 검사
isgraph(char)	그래픽 문자 검사
iscntrl(char)	제어문자('\a', '\b', '\n', '\t', '\f', '\v', '\r') 검사
toupper(char)	영문 소문자를 대문자로 변환
tolower(char)	영문 대문자를 소문자로 변환
toascii(char)	아스키코드로 변환
_tolower(char)	무조건 영문 소문자로 변환
_toupper(char)	무조건 영문 대문자로 변환

실습예제 9-10 Prj10 10char.c 다양한 문자 관련 함수 난이도: ★

```

01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 #include <ctype.h> //문자 관련 함수는 헤더파일 ctype.h에 매크로로 정의
04
05 void print2char(char);
06
07 int main(void)
08 {
09     char ch;
10
11     printf("알파벳(종료x) 또는 다른 문자 입력하세요.\n");
12     do
13     {
14         printf("문자 입력 후 Enter: ");
15         scanf("%c", &ch);
16         getchar(); //문자 하나를 입력한 후 enter키를 반드시 누르도록 해 이 enter키를 하나 받아 들어 버리는 기능
17         if (isalpha(ch)) //enter 기 입력 받음
18             print2char(ch); //입력 문자 ch가 영문자 알파벳 여부 검사
19         else
20             printf("입력: %c\n", ch); //입력 문자가 영문자 알파벳이 아니면 바로 문자 그대로 출력
21     } while (ch != 'x' && ch != 'X'); //입력이 x 또는 X으면 종료
22
23     return 0;
24 }
25
26 void print2char(char ch)
27 {
28     if (isupper(ch))
29         printf("입력: %c, 변환: %c\n", ch, tolower(ch));
30     else
31         printf("입력: %c, 변환: %c\n", ch, toupper(ch));
32
33     return;
34 }

결과
알파벳(종료x) 또는 다른 문자 입력하세요.
문자 입력 후 Enter: w
입력: w, 변환: W
문자 입력 후 Enter: t
입력: t, 변환: T
문자 입력 후 Enter: e
입력: e
문자 입력 후 Enter: x
입력: x, 변환: X

```

다양한 라이브러리 함수를 제공

- 여러 헤더파일이 제공
 - 여러 라이브러리 함수를 위한 함수원형과 상수, 그리고 매크로가 여러 헤더파일에 나뉘어 있음

표 9-3 여러 라이브러리를 위한 헤더파일

헤더파일	처리 작업
stdio.h	표준 입출력 작업
math.h	수학 관련 작업
string.h	문자열 작업
time.h	시간 작업
ctype.h	문자 관련 작업
stdlib.h	여러 유ти리티(텍스트를 수로 변환, 난수, 메모리 할당 등) 함수