<6장> 함수

학습 목표

- 함수의 필요성을 이해한다.
- 함수를 정의하고 구현하는 방법을 확인한다.
- 변수의 종류(지역, 전역변수)와 유효 범위를 이해한다.
- 함수 선언의 필요성과 선언 방법을 알아본다.
- 재귀 호출을 학습한다.
- 매크로 함수를 사용해 본다.
- 함수를 왜 사용하는지, 어떻게 구현하는지, 함수 내부 또는 외부에서 변수를 선언했을 때의 차이 등을 알아본다.

목차

01

함수

1. 함수

■ 함수의 사용 목적

▪ C 언어의 함수 사용 목적

- 전체를 함수로 부품화
- 함수 단위로 구성하여 가독성 향상
- 반복적으로 사용되는 코드를 묶어서 재사용성 향상
- 함수 단위로 코드를 검수할 수 있어 안전성 향상
- 함수의 인터페이스는 변경하지 않고, 내부를 교체하기 용이
- 부품 단위로 나누어져서 협업하기 용이. 함수의 인터페이스만 정해지면 각자 나누어 구현 가능

1. 함수

캡슐화와 재사용성

- 캡슐화
  - 내부 동작 원리 및 구조를 밖에 드러내지 않고 사용하는데 필요한 인터페이스만 제공하는 것

- 재사용성
  - 재사용성을 높이기 위해 함수는 단순화시키고 일반화해서 작성

02

함수 정의, 호출, 실행

2. 함수 정의, 호출, 실행

- 함수 정의(define) 방법

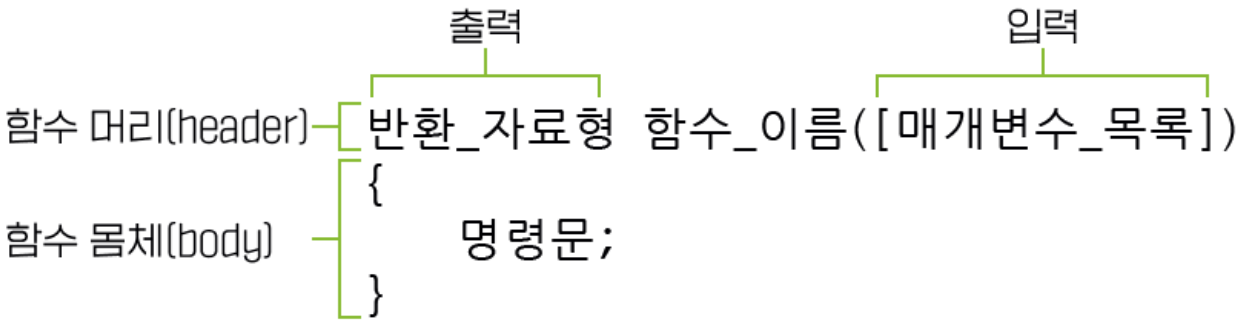


그림 6-1 함수 정의 방법

- 함수 구현에 필요한 템플릿

코드 6-1

```
1   Type func(Type param1, Type param2)
2   {
3       S1;
4   }
```

2. 함수 정의, 호출, 실행

함수 코드 실행 후 반환하는 값

- return 문
  - 함수 몸체 중간에서 실행 종료 가능
  - 값을 반환할 수 있음

```
return 표현식;  // (1)
return ;        // (2)
```

2. 함수 정의, 호출, 실행

함수 호출

- 함수 이름에 괄호를 붙이고, 매개변수에 전달할 값이 있다면 괄호 안에 나열

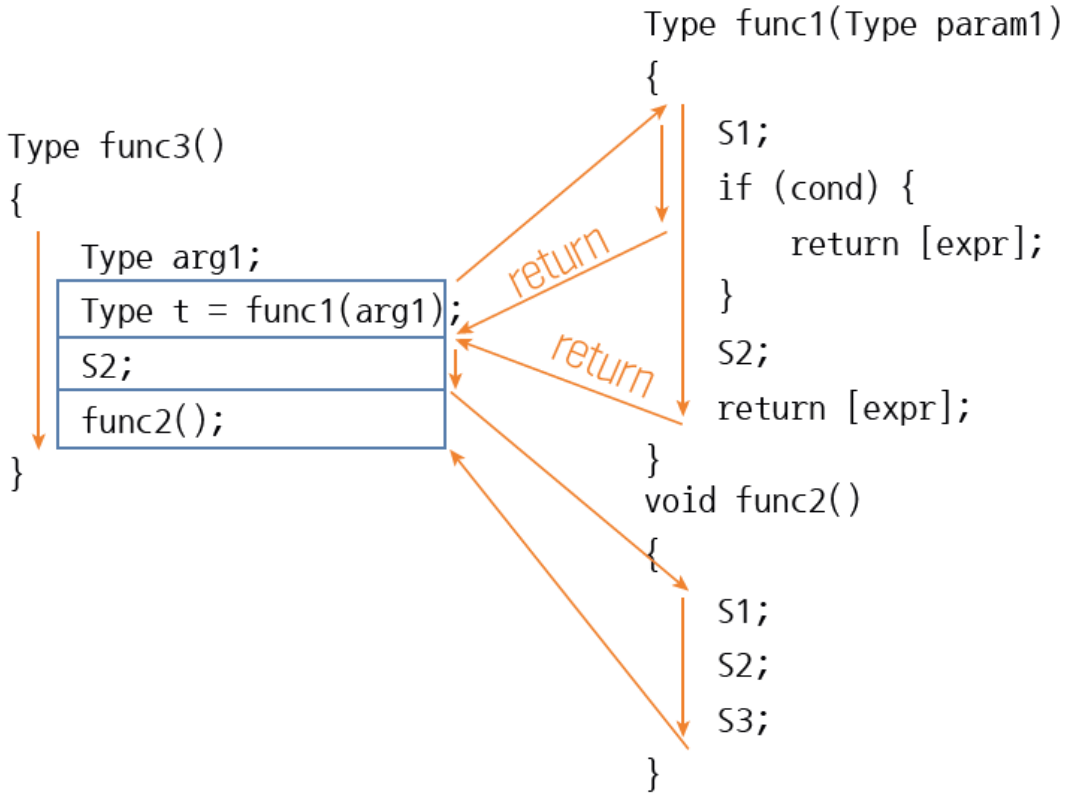


그림 6-2 함수 호출 및 실행 과정

2. 함수 정의, 호출, 실행

■ 함수의 매개변수에 인자 전달

- 함수를 호출할 때 인자가 매개변수에 전달되는 과정

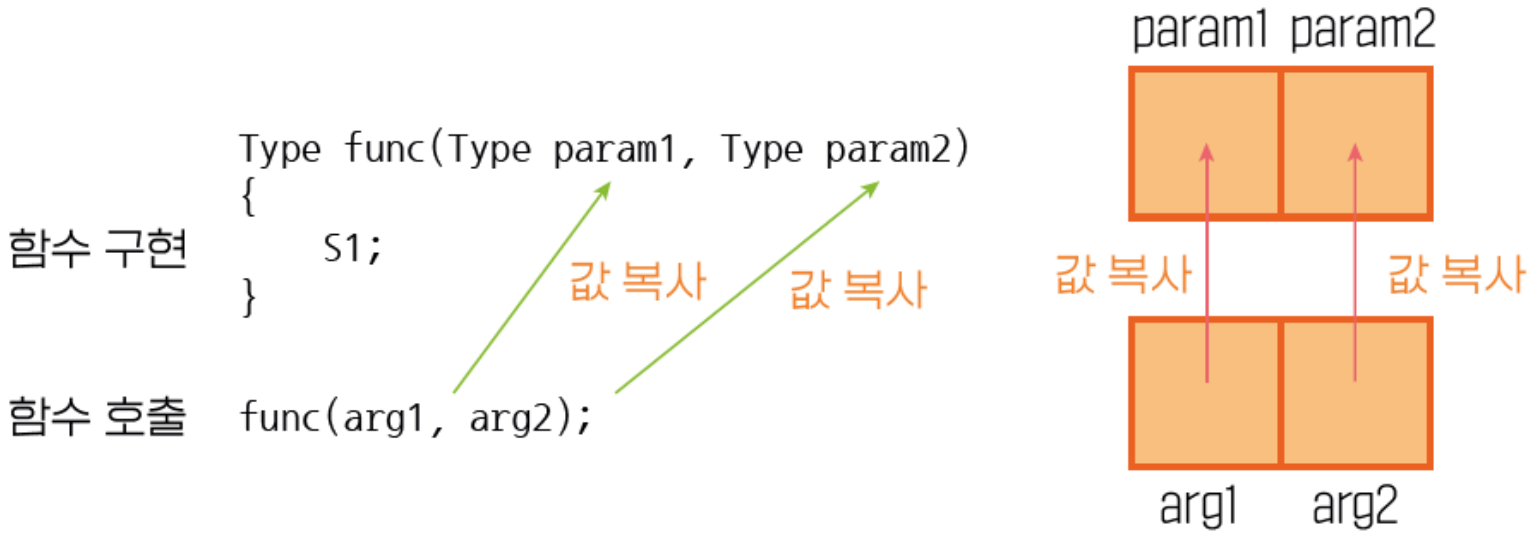


그림 6-3 함수 호출 시 인자가 매개변수에 전달되는 과정

2. 함수 정의, 호출, 실행

■ 함수의 매개변수에 인자 전달

- 인자를 매개변수에 값에 의한 호출 방식으로 전달

코드 6-2

```
1    Type param1 = arg1; // 인자를 매개변수에 복사
2    Type param2 = arg2; // 인자를 매개변수에 복사
3    S1;
```

▪ C 언어의 값 전달 방식 특징

- 인자에 표현식을 사용 가능(변수나 리터럴 상수도 표현식 중 한 가지)
- 함수에서 매개변수의 값을 변경해도 함수 바깥에 영향을 주지 않음

주어진 시간을 1분 증가시켜 변경되는 시간 정보를 출력하는 프로그램을 만들자

```
#define  _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
// 시간을 1분 증가시키는 함수 (값 반환)
int addOneMinute(int hour, int minute) {
    minute++;
    if (minute == 60) {
        minute = 0;
        hour++;
        if (hour == 24) {
            hour = 0;
        }
    }
    return hour * 100 + minute;
     // 시간과 분을 하나의 정수로 조합하여 반환
}
```

```
int main() {
    int hour, minute, time;

    // 사용자로부터 시간 입력 받기
    printf("시간과 분을 입력하세요 (예:12 30): ");
    scanf("%d %d", &hour, &minute);

    // 1분 증가 함수 호출 및 결과 저장
    time = addOneMinute(hour, minute);

    // 결과 출력 (시간과 분을 다시 분리)
    printf("변경된 시간은 %02d:%02d입니다.₩n", time / 100, time % 100);

    return 0;
}
```

```
시간과 분을 입력하세요 (예:12 30): 03 20
변경된 시간은 03:21입니다.
```

2. 함수 정의, 호출, 실행

■ 함수의 매개변수에 인자 전달

▪ swap() 함수 구현 문제

- 두 개의 매개변수로 전달된 값을 서로 바꿔 함수 밖에 전달하는 swap() 함수

코드 6-4 Swap.c

```
1   #include <stdio.h>
2
3   void swap(int num1, int num2)
4   {
5       int temp = num1;
6       num1 = num2;
7       num2 = temp;
8   }
9    int main(void)
10  {
11      int x = 5;
12      int y = 9;
13      swap(x, y); // 함수 호출 후 x에는 9가 y에는 5가 있을 거라고 예상
14      printf("x = %d, y = %d\n", x, y);
15      return 0;
16  }
```

swap() 함수 구현
num1을 임시 변수에 저장하고 num2를 num1에 저장
임시 변수의 값을 num2에 저장해서 서로 교환

swap() 함수에 전달해서 서로 교체할 x, y 변수 선언 및 초기화

swap() 함수 호출

값이 서로 바뀌었는지 확인하기 위해 x, y 출력

〈실행 결과〉

```
x = 5, y = 9
```

2. 함수 정의, 호출, 실행

■ 함수의 매개변수에 인자 전달

- swap() 함수 구현 문제

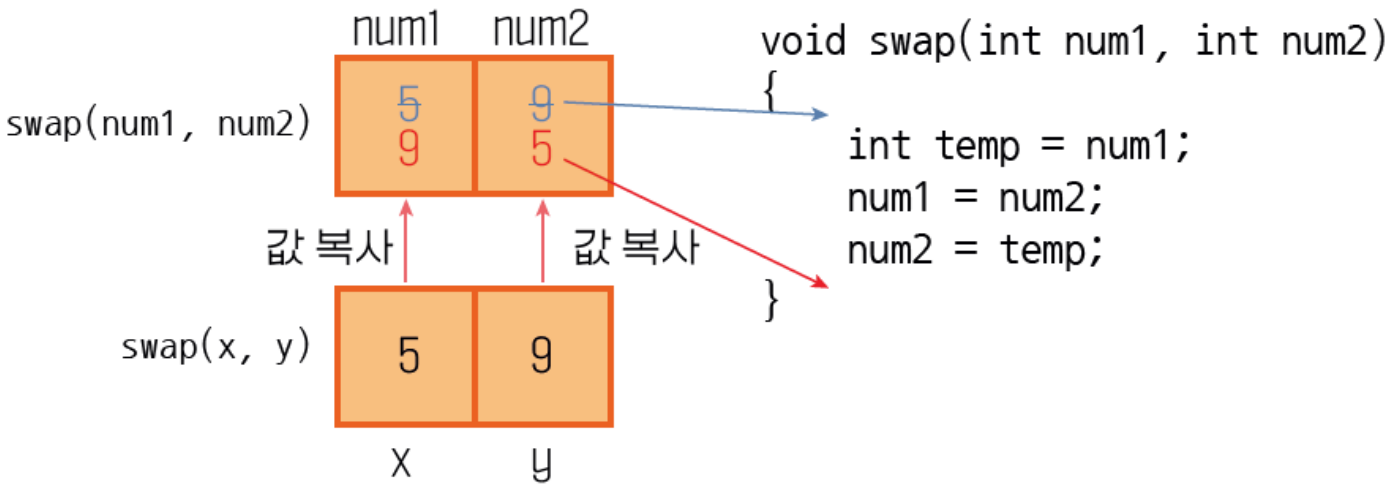


그림 6-4 swap() 함수에 매개변수가 전달되어 동작하는 방법

2. 함수 정의, 호출, 실행

다시 살펴보는 main() 함수

- <1장>에서 처음으로 작성했던 프로그램 코드의 일부

코드 6-5

```
1   int main(void)
2   {
3       printf("My first C program"); // 문자열 출력하는 printf() 함수 호출
4       return 0;
5   }
```

- 함수의 반환_자료형은 int, 이름은 main, 매개변수는 없으므로 void로 지정
- 함수가 호출되면 실행되는 코드의 시작. 화면에 주어진 문자열을 출력
- 함수 코드 실행을 종료시키면서 main() 함수의 결괏값으로 0 반환
  return 문이 없었어도 함수는 더 이상 실행시킬 코드가 없으므로 종료
  return 문은 함수 실행을 종료하는 것보다는 0을 반환하는 것이 실행 목적

2. 함수 정의, 호출, 실행

입력으로 전달된 양의 정수가 소수인지 확인하는 함수

- 표 6-1을 참고하여 입력을 매개변수로 전달받는 함수 구현

표 6-1 소수인지 확인하는 함수의 속성

반환 자료형	함수 이름	매개변수
int	isPrimeNumber	int

- 알고리즘

```
변수 i를 2~m-1까지의 정수를 한 개씩 저장하며 반복
    m을 i로 나눈 나머지가 0이면
        약수가 발견되었으므로 0을 반환
반복이 종료되면 약수가 발견되지 않았다는 것이므로 소수임을 나타내는 1을 반환
```

2. 함수 정의, 호출, 실행

입력으로 전달된 양의 정수가 소수인지 확인하는 함수

- 표와 알고리즘에 맞춰 함수 구현

코드 6-6

```
1   int isPrimeNumber(int m)
2   {
3       for (int i = 2; i < m; i++) { // 2~m까지의 정수로 m이 나눠지는지 검사
4           if (m % i == 0) { // 약수
5               return 0; // 약수가 있으므로 소수가 아님. 0 반환
6           }
7       }
8       return 1; // 반복문이 종료되었다는 것은 약수가 없다는 뜻. 1 반환
9   }
```

2. 함수 정의, 호출, 실행

■ 입력으로 전달된 양의 정수가 소수인지 확인하는 함수

- 0과 1에 대해서 처리하는 코드 포함

코드 6-7

```
1   int isPrimeNumber(int m)
2   {
3       if (m < 2) { return 0; }
4       else if (m == 2) { return 1; }
5       for (int i = 3; i < m; i+=2) { // 3~m까지의 홀수로 m이 나눠지는지 검사
6           if (m % i == 0) { // 약수
7               return 0; // 약수가 있으므로 소수가 아님. 0 반환
8           }
9       }
10      return 1; // 반복문이 종료되었다는 것은 약수가 없다는 뜻임. 1 반환.
11  }
```

※ 입력으로 전달된 양의 정수가 소수인지 확인하는 함수 완성

➔ 코드 6-8
➔ 실행 결과

2. 함수 정의, 호출, 실행

n1~n2 사이의 정수 중 소수를 출력하는 함수

함수의 속성

표 6-2 소수인지 확인하는 함수의 속성

반환 자료형	함수 이름	매개변수
void	printPrimeNumbers	int, int

함수 구현

코드 6-9

```
1   // 주어진 n1~n2 범위 내의 소수를 출력하는 함수
2   void printPrimeNumbers(int n1, int n2) {
3       // 주어진 범위에서 소수인지 검사하고 출력
4       for (int i = n1; i <= n2; i++) {
5           if (isPrimeNumber(i)) { printf("%d\n ", i); }
6       }
7   }
```

2. 함수 정의, 호출, 실행

n1~n2 사이의 정수 중 소수를 출력하는 함수

- 알고리즘

코드 6-10

```
1   void printPrimeNumbers(int n1, int n2)
2   {
3       if (n1 >= 2) {
4           if (n1 == 2 || n2 == 2) { printf("소수: 2\n"); }
5           if (n1 % 2 == 0) { n1++; }
6           for (int m = n1; m <= n2; m+=2) {
7               if (isPrimeNumber(m)) { printf("소수: %d\n", m); }
8           }
9       }
10  }
```

printPrimeNumbers() 함수 정의 시작

n1이 2 이상일 때만 알고리즘 적용

n1~n2까지 홀수만 소수인지 확인. n1은 홀수

소수면 출력

※ n1~n2 사이의 정수 중 소수를 출력하는 함수 완성

➔ 코드 6-11
➔ 실행 결과

03
변수의 유효 범위와 생존 기간

3. 변수의 유효 범위와 생존 기간

유효 범위(scope)

- C 언어 코드를 변수들이 선언된 위치를 통해 그 변수가 어디서 사용할 수 있는지 확인하는 것

변수의 유효 범위

- 전역
- 지역
- 코드 블록
- for 문
- 파일

3. 변수의 유효 범위와 생존 기간

지역변수

- 함수 내부에서 선언되고 그 함수에서만 사용할 수 있는 변수
- 매개변수도 지역변수로 함수 내에서 동일한 명칭으로 변수 선언 불가

```
int add(int x, int y)
{
    int result = x + y;
    return result;
}

int subtract(int x, int y)
{
    int result = x - y;
    return result;
}
```

add() 함수의 x, y, result 사용 가능 영역 (add() 함수 유효 범위)

subtract() 함수의 x, y, result 사용 가능 영역 (subtract() 함수 유효 범위)

그림 6-5 지역변수의 유효 범위

3. 변수의 유효 범위와 생존 기간

■ 지역변수

- 근의 공식을 이용해서 이차 방정식의 해를 구하는 프로그램

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

※ 지역변수 코드 완성

➔ 코드 6-12
➔ 실행 결과

3. 변수의 유효 범위와 생존 기간

전역변수

- 함수 밖에 선언하고 프로그램의 모든 함수에서 사용할 수 있는 변수
- 프로그램 전체에 영향을 미치기 때문에 동일한 이름의 변수 선언 불가

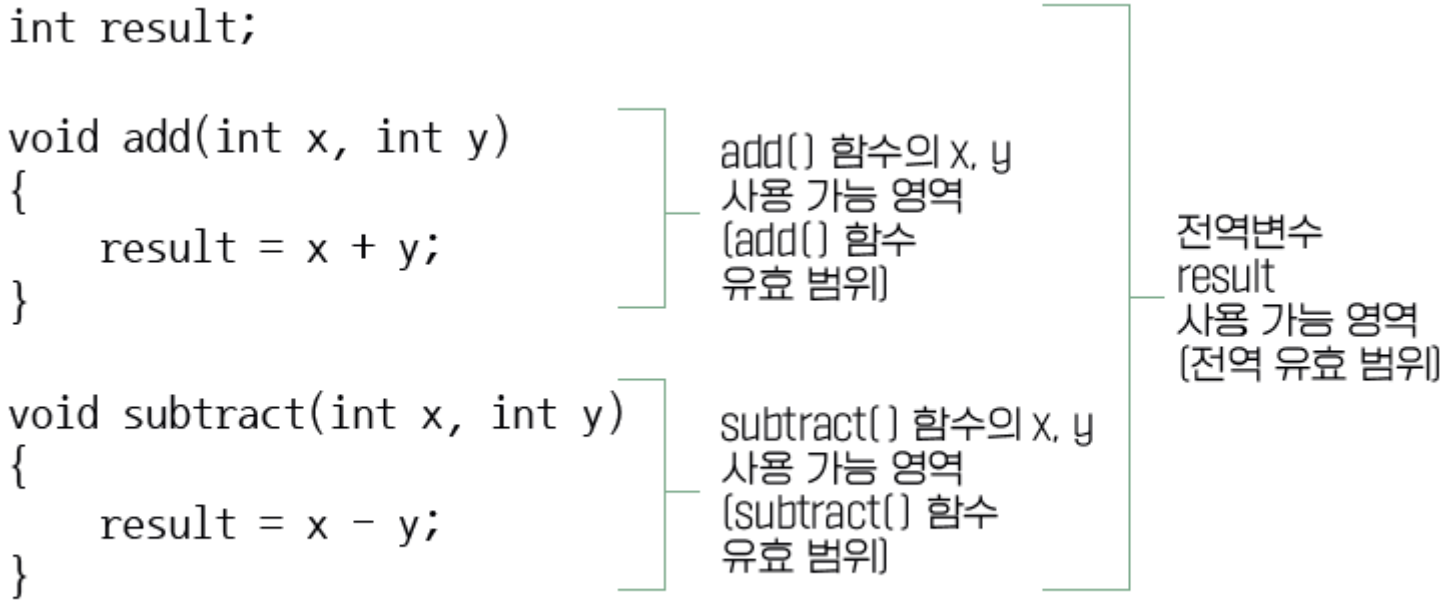


그림 6-6 지역변수와 전역변수의 유효 범위

※ 전역변수에 값을 저장하고 main() 함수에서 출력하는 코드

➔ 코드 6-13
➔ 실행 결과

3. 변수의 유효 범위와 생존 기간

전역변수

- 전역변수를 이용해서 무작위로 정수를 생성하는 함수
  - 임의로 만든 간단한 수학 공식을 이용해 난수를 생성하는 프로그램 작성

$$seed = seed^2 + 13 * seed + 19$$

  - 함수 원형

```
void setSeed(int s);  // 시드 값을 지정
int random( ); // 0~RAND_MAX의 정수를 생성해서 반환
```

  - 변수 선언

```
unsigned seed = 17;
```

3. 변수의 유효 범위와 생존 기간

전역변수

- 전역변수를 이용해서 무작위로 정수를 생성하는 함수
  - setSeed() 함수

```
void setSeed(unsigned s)
{
    seed = s;
}
```

  - random() 함수

```
unsigned random()
{
    seed = (seed * seed) + (13 * seed) + 19;
    if (seed > RAND_MAX) { seed %= RAND_MAX; }
    return seed;
}
```

난수 발생 공식에 전역변수 seed를 적용하고 결과를 다시 seed에 저장

seed가 너무 큰 경우 범위 안의 정수로 변환

※ 전역변수에 값을 저장하고 main() 함수에서 출력하는 코드

➔ 코드 6-14
➔ 실행 결과

3. 변수의 유효 범위와 생존 기간

■ 전역변수와 지역변수의 이름 충돌과 유효 범위

- 같은 이름의 변수들을 다른 함수들에 사용하는 것은 괜찮음

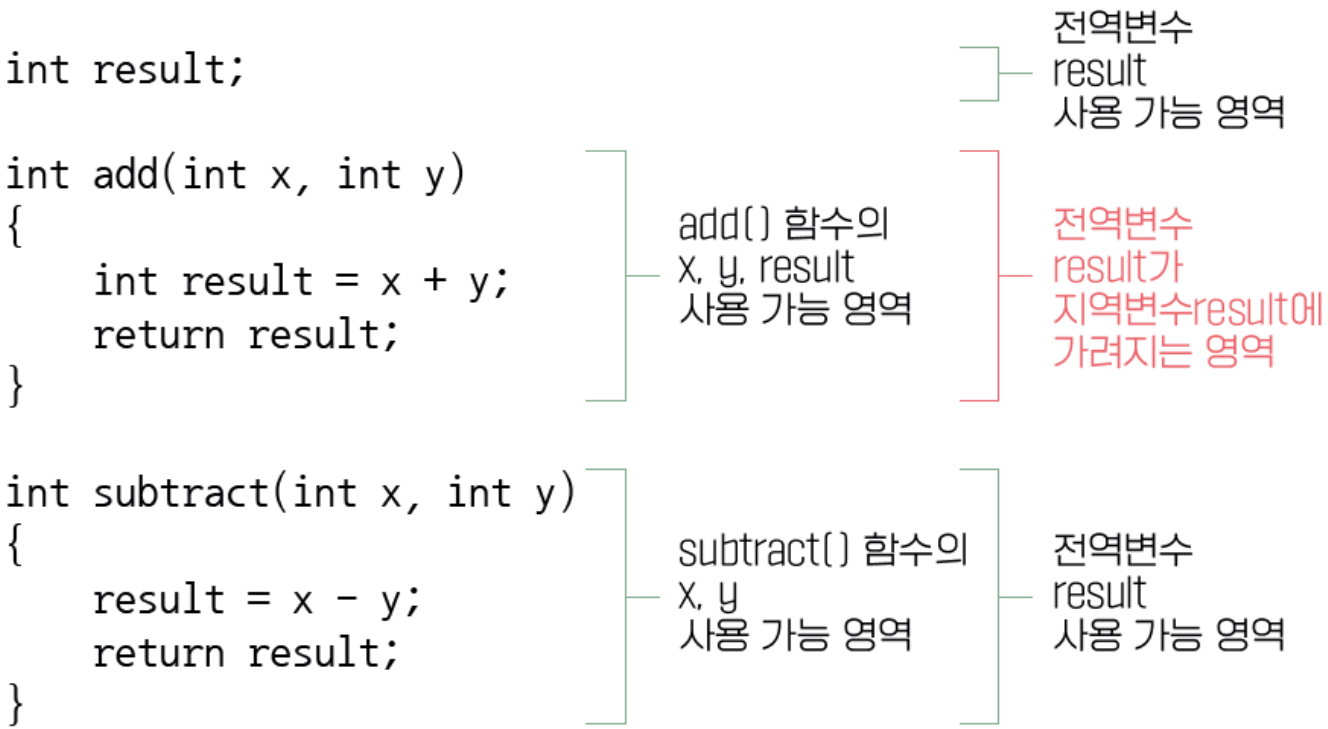


그림 6-7 동일한 이름의 변수가 사용되는 경우 좁은 유효 범위를 가지는 변수를 사용

※ 전역변수와 지역변수의 이름 충돌과 유효 범위 코드

➔ 코드 6-16
➔ 실행 결과

3. 변수의 유효 범위와 생존 기간

코드 블록과 for 문 유효 범위

- 코드 블록은 자신만의 유효 범위를 가짐

```
int x = 5;

int main(void)
{
    printf("x = %d\n", x);
    int x = 7;
    printf("x = %d\n", x);
    {
        printf("x = %d\n", x);
        int x = 9;
        printf("x = %d\n", x);
    }
    for (int x = 0; x < 2; x++) {
        printf("x = %d\n", x);
    }
    printf("x = %d\n", x);
}
```

전역변수 x의 사용 영역

main() 함수에 선언된 지역변수 x 의 사용 영역

코드 블록에 선언된 변수 x 의 변수 사용 영역

for 문에 선언된 변수x 의 사용 영역

main() 함수에 선언된 지역변수 x 의 사용 영역

그림 6-8 동일한 이름의 변수가 사용되는 경우 좁은 유효 범위를 가지는 변수 사용

※ 코드 블록과 for 문 유효 범위 코드

➔ 코드 6-17
➔ 실행 결과

3. 변수의 유효 범위와 생존 기간

정적 지역변수

- 유효 범위는 지역변수와 동일
- 생존 기간은 전역변수와 일치하는 변수
- 함수가 종료되어도 정적 지역변수가 사용하는 메모리 공간은 변함없이 존재

코드 6-16

```
Type func( )
{
    static Type v1 = 초깃값;  // 정적 지역변수 선언
    Type v2;                  // 지역변수 선언
}
```

- 정적 지역변수의 두 가지 특징
  - 함수가 여러 번 호출되더라도 초기화 코드는 실행되지 않음
  - 정적 지역변수를 초기화하는 것은 상수로만 가능

※ 정적 지역변수 코드

➔ 코드 6-19
➔ 실행 결과

3. 변수의 유효 범위와 생존 기간

typedef와 유효 범위

- 유효 범위 설명
  - 함수 밖에 선언된 자료형 이름은 현재 선언된 소스 파일 안에서 유효
  - 함수 안에서 선언된 이름은 함수에서 유효
  - 코드 블록 안에서 선언된 자료형 이름은 코드 블록 안에서 유효

※ 월을 나타내는 enum 자료형을 만들고
사용자로부터 정수를 입력받아 eunm 값으로
반환하는 함수 코드

➔ 코드 6-20
➔ 실행 결과

3. 변수의 유효 범위와 생존 기간

다시 살펴보는 const 변수

- 매개변수를 이용해 const 변수 초기화

코드 6-21 const.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   void func(int n)
5   {
6       const int m = n;   ← 매개변수로 전달된 값을 이용해서 const 변수 초기화
7       const int v = 3; // 당연히 상수로도 초기화 가능
8       printf("m = %d\n", m);
9   }
10
11  int main(void)
12  {
13      func(50);
14      return 0;
15  }
```

<실행 결과>

```
m = 50
```

04

함수 선언

4. 함수 선언

- 코드 6-11의 함수 순서를 변경하여 main() 함수를 앞에 위치시키기

코드 6-22 QuadEqnLocal2.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3   #include <math.h>
4
5   int main(void)
6   {
7       printQuadEqnSoln(1, -5, 6);
8       printQuadEqnSoln(1, 5, -6);
9       return 0;
10  }
11
12  void printQuadEqnSoln(int a, int b, int c)
13  {
14      double r = sqrt(b * b - 4 * a * c);
15      double x1 = (-b + r) / (2 * a);
16      double x2 = (-b - r) / (2 * a);
17      printf("x1 = %f, x2 = %f\n", x1, x2);
18  }
```

4. 함수 선언

- 함수 선언은 반드시 함수의 머리 부분에 세미콜론을 붙여서 문장 생성

```
반환_자료형 함수_이름(매개변수_목록);
```

- 코드 6-20의 함수 원형

```
void printQuadEqnSoln(int a, int b, int c);
```

- 코드 6-20의 함수 원형의 다른 형태

```
void printQuadEqnSoln(int, int, int);
```

※ 함수 원형을 포함해서 경고나 오류 없이
컴파일되도록 만든 코드

➔ 코드 6-23

05

표준 C 라이브러리 헤더 파일

5. 표준 C 라이브러리 헤더 파일

- ANSI C에서 제공하는 표준 C 라이브러리에서 자주 사용하는 헤더 파일

표 6-3 자주 사용하는 표준 헤더(많이 사용하는 함수들이나 자료형 등을 선언하는 헤더 파일들)

표준 헤더 파일 이름	설명
ctype.h	문자 관련 함수들이 포함된다.
float.h	컴파일러에서 정의하는 실수 자료형 관련 내용(예 : 범위)들이 포함된다.
limits.h	컴파일러에서 정의하는 정수 자료형 관련 내용(예 : 범위)들이 포함된다.
math.h	수학 관련 함수들이 포함된다.
stdio.h	입출력 관련 함수들이 포함된다.
stdlib.h	문자열과 숫자 변환 함수들, 메모리 할당 함수들, 정렬 함수 등 여러 가지 기능들을 담당하는 함수들을 모아놓았다.
string.h	문자열 처리 함수들이 포함된다.
stddef.h	NULL, size_t 등이 포함된다.
time.h	시간 관련 함수들이 포함된다.

06

메모리 구조

6. 메모리 구조

- 프로그램의 메모리 구조

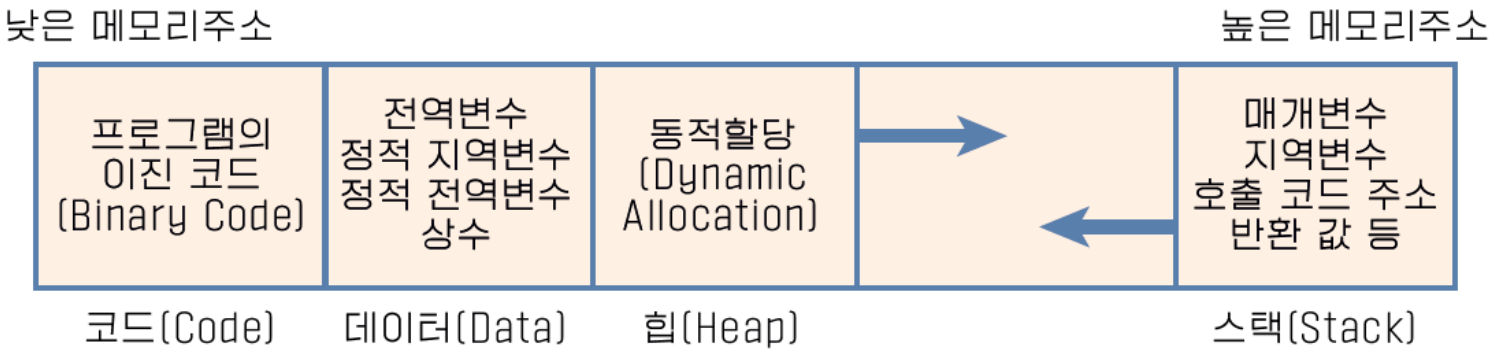


그림 6-9 운영체제에서 실행한 프로그램의 메모리 구조

- 코드 영역(텍스트 영역)
- 데이터 영역
- 힙 영역
- 스택 영역

6. 메모리 구조

- 프로그램의 메모리 구조
  - func1()의 printf() 함수 내부 코드가 실행되고 있을 때의 스택 구조

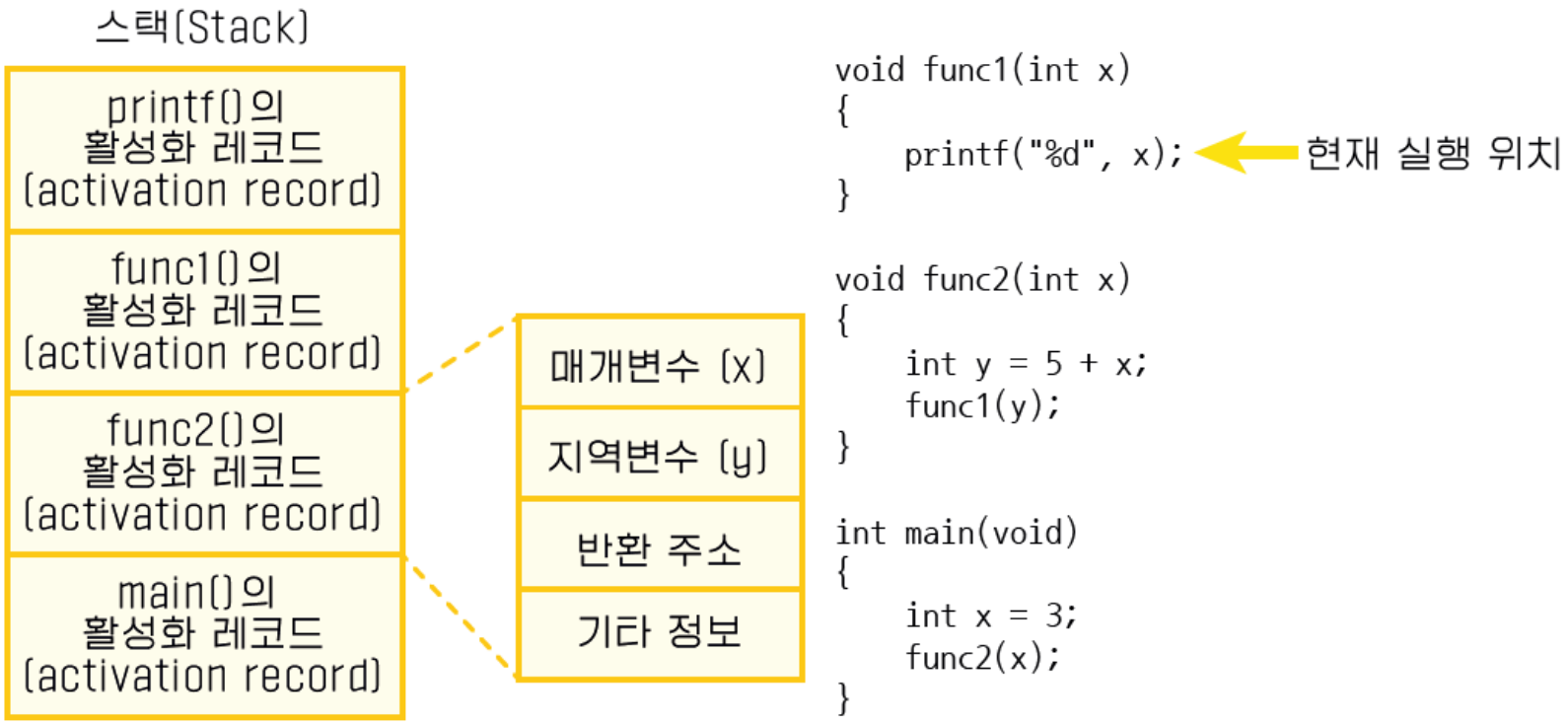


그림 6-10 운영체제에서 실행한 프로그램의 메모리 구조

6. 메모리 구조

- 프로그램의 메모리 구조
  - func1()의 printf() 함수가 종료되고, func1()의 함수가 종료되기 직전의 스택

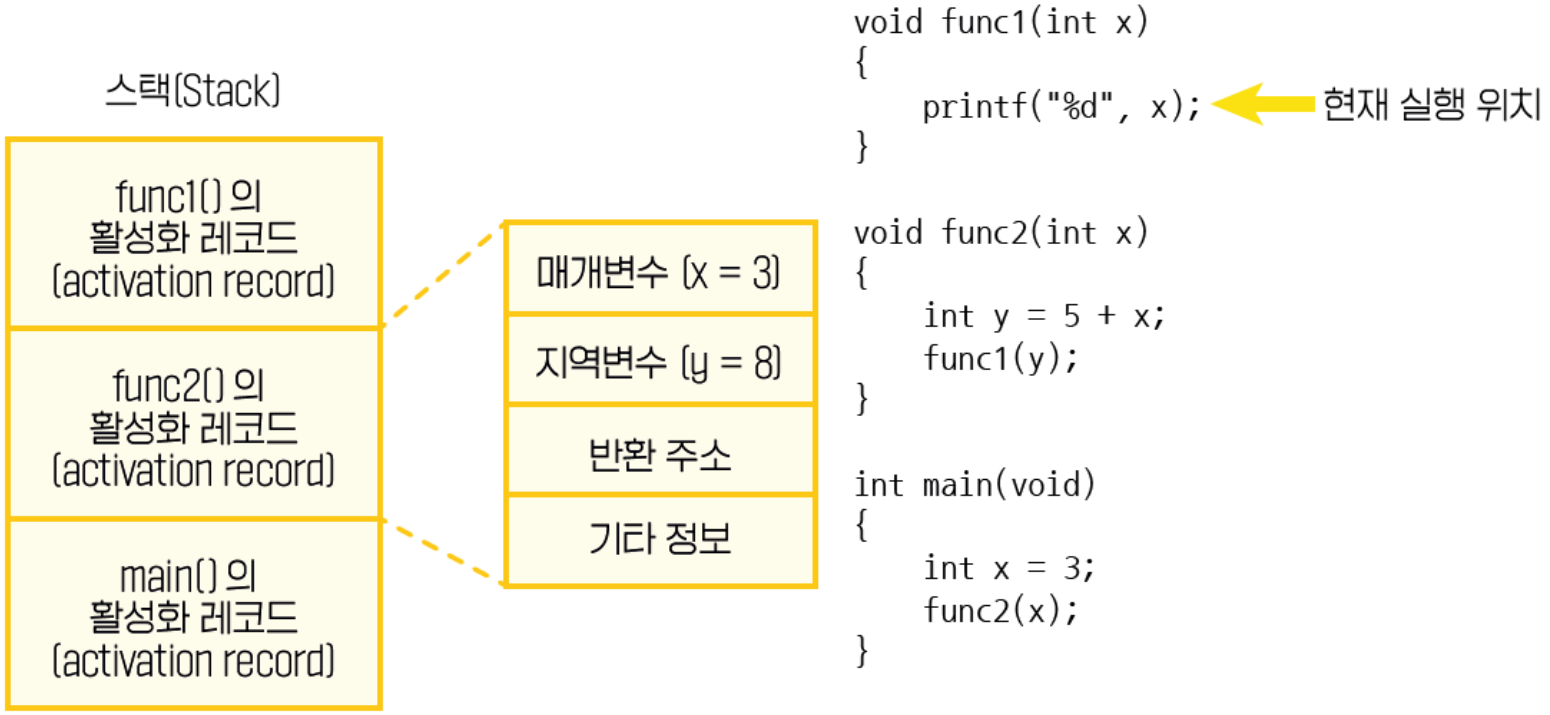
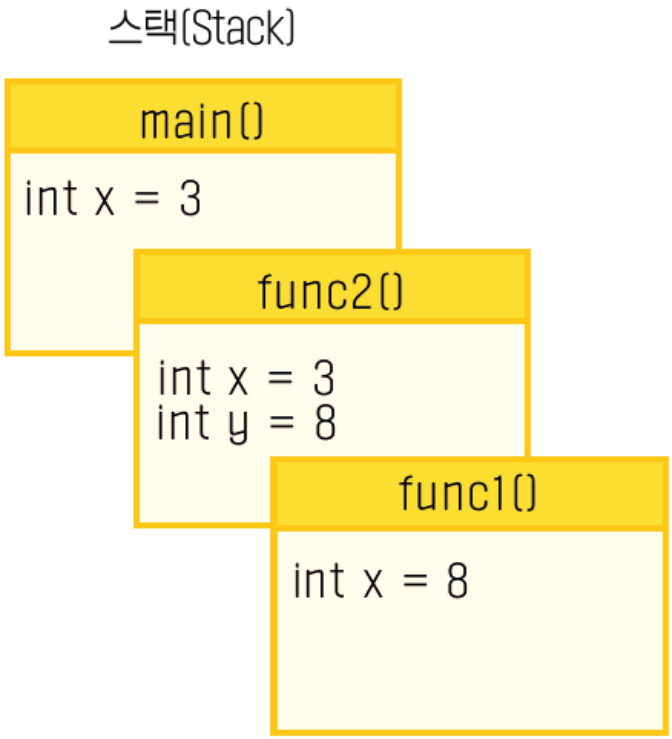


그림 6-11 운영체제에서 실행한 프로그램의 메모리 구조

6. 메모리 구조

- 프로그램의 메모리 구조
  - 메모지를 이용해서 스택을 직접 구성해 보기




```
void func1(int x)
{
    printf("%d", x);
}

void func2(int x)
{
    int y = 5 + x;
    func1(y);
}

int main(void)
{
    int x = 3;
    func2(x);
}
```

현재 실행 위치

그림 6-12 메모지를 이용해서 확인하는 스택 구조

07

재귀 호출

7. 재귀 호출

- 재귀(recursion)는 문제를 같은 유형의 작은 문제로 나누어서 해결하는 방법

재귀적 알고리즘

1 + 2 + … + n을 계산하는 문제

- n이 1 이상의 양의 정수라고 가정
- 함수 f(n)을 1부터 n까지의 정수의 합이라고 정의

```
f(n) = 1 + 2 + ... + (n - 1) + n          (n >= 1)
```

- 함수 f(n)의 n에 1, 5, 10을 전달해서 계산

```
f(1) = 1
f(5) = 1 + 2 + 3 + 4 + 5 = 15
f(10) = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55
```

7. 재귀 호출

재귀적 알고리즘

- 1 + 2 + … + n을 계산하는 문제
  - 자기 자신을 이용해서 f(n)을 다시 정의

코드 6-24

```
f(n) = f(n - 1) + n          (n >= 1)
f(n) = 0                     (n == 0)
```

- 재귀 호출 코드

코드 6-25

```
1    int f(int n)
2    {
3        if (n == 0) { return 0; }
4        return f(n - 1) + n;
5    }
```

만약 n == 0이면 0을 반환

n이 1 이상이라면 f(n - 1) + n을 반환
f() 함수는 자기 자신을 다시 호출하는 코드를 포함

※ 코드 6-25를 이용해서 합을 구하는 프로그램

➔ 코드 6-26
➔ 실행 결과

7. 재귀 호출

■ 재귀 함수의 동작 원리

- 코드 6-26의 sum1toN() 함수가 동작하는 방법

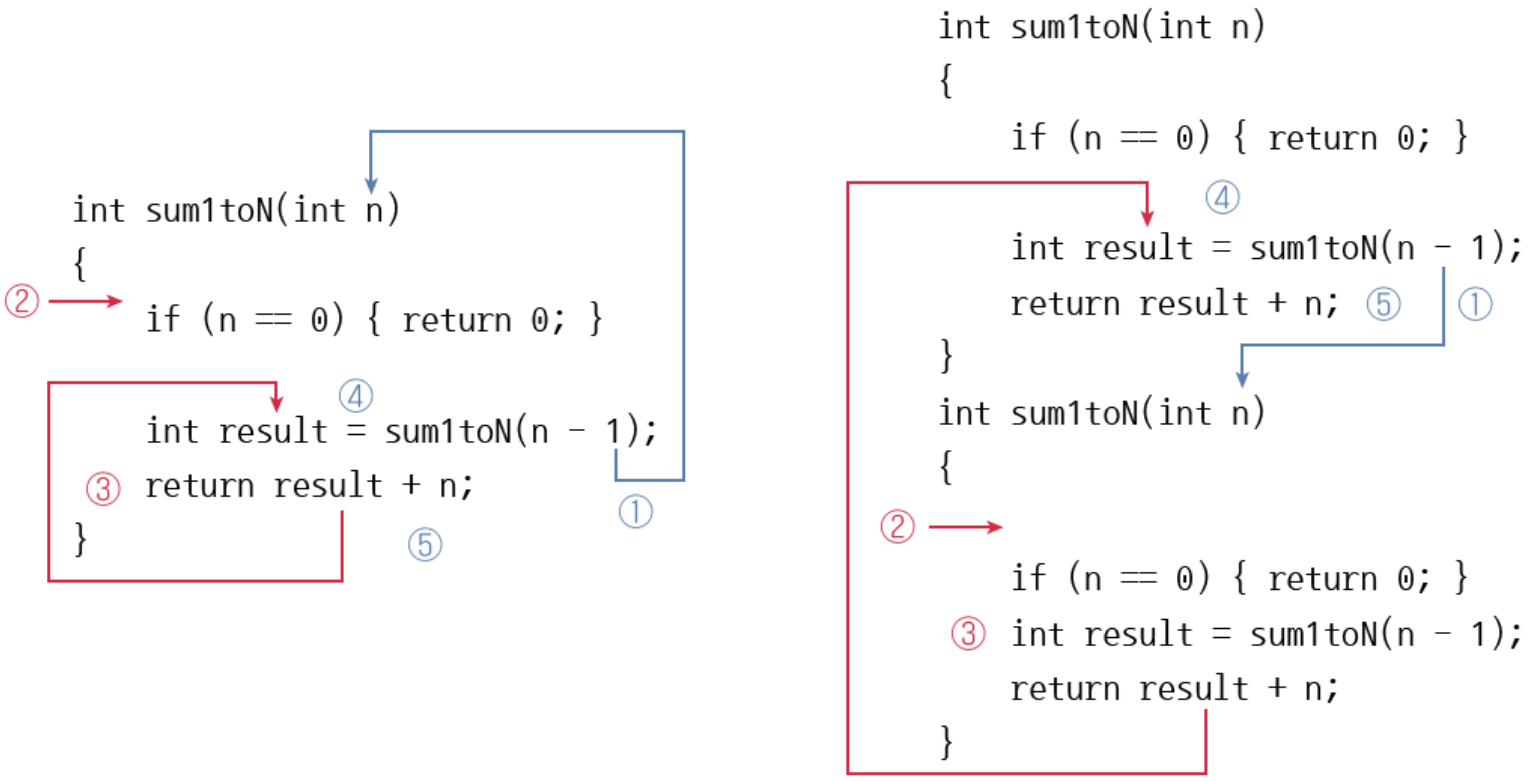


그림 6-13 재귀 함수 sum1toN()이 동작하는 방법

7. 재귀 호출

반복문과 재귀 호출

- 반복문으로 해결하는 문제는 모두 재귀 호출을 이용해서 해결 가능
- 그 반대로 재귀 호출을 이용해서 해결할 수 있는 문제는 반복문으로도 구현 가능

코드 6-27

```
1    int f(int n)
2    {
3        int sum = 0;
4        for (int i = 0; i <= n; i++) {
5            sum += i;
6        }
7        return sum;
8    }
```

08

매크로 함수

8. 매크로 함수

- 매크로 함수 선언

```
#define 매크로_이름(인자1, 인자2, ...) 치환할_값
```

- 매크로 함수 선언 주의점
  - 인자 개수는 제한 없음
  - 매크로_이름과 괄호 사이에 공백이 있으면 안 됨
  - 치환할_값은 인자1, 인자2 등을 포함하지 않아도 되지만, 대부분 포함하는 코드로 구성
  - 치환할_값은 (인자1), (인자2)와 같이 괄호로 감싼 형태로 사용
  - 매크로 함수 사용은 매크로_이름(표현식1, 표현식2, ...) 형태로 사용
  - 매크로 함수를 여러 줄로 표현하려면 마지막 줄을 제외한 각 줄의 끝에 \ 를 붙임

8. 매크로 함수

- 매크로 함수 선언과 사용 예시

코드 6-28

```
1    #define PRINT(msg)    printf((msg))
2    #define MUL(n1, n2)   ((n1) * (n2))
```

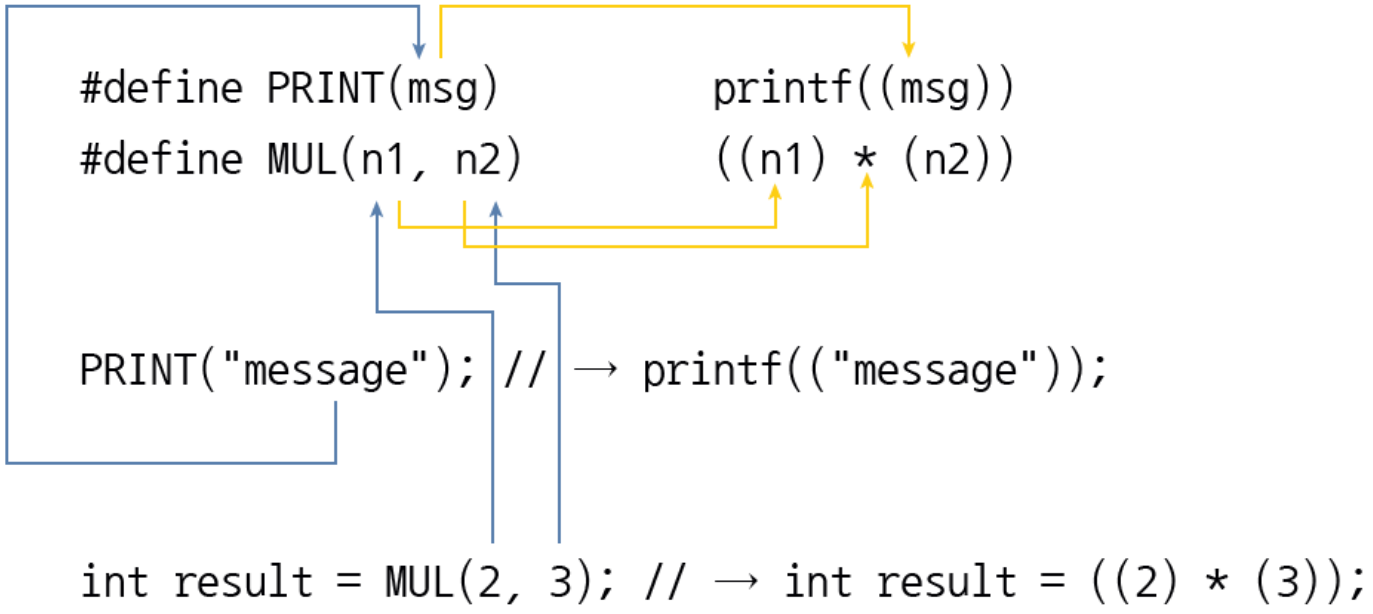


그림 6-17 매크로 함수 선언 및 사용

8. 매크로 함수

- 매크로 함수 선언과 사용 예시
  - MUL()에 2와 3 + 4, 즉 2와 7을 전달하고 14를 result에 저장할 것이라고 예상

코드 6-29

```
int result = MUL(2, 3 + 4); --> int result = ((2) * (3 + 4));
```

  - 코드의 변수나 다른 표현식을 인자로 전달

```
int n = 4;
int result = MUL(2, n + 3); // --> int result = ((2) * (n + 3)) ;
```

  - 인자를 감싸는 괄호가 없을 때

```
int result = MUL(2, 3 + 4); --> int result = 2 * 3 + 4; // 괄호가 없어서 10이 됨
```

8. 매크로 함수

- SWAP_INT() 함수를 매크로로 구현

코드 6-30 SwapInt.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   #define SWAP_INT(x, y) int t = (x); (x) = (y); (y) = t;
5
6   int main(void)
7   {
8       int a = 5;
9       int b = 4;
10      printf("Before SWAP: a = %d, b = %d\n", a, b);
11      SWAP_INT(a, b);
12      printf("After SWAP: a = %d, b = %d\n", a, b);
13      return 0;
14  }
```

두 개 인자를 교환하는 코드 작성

매크로 함수를 이용해서 a와 b의 값을 교환

<실행 결과>

```
Before SWAP: a = 5, b = 4
After SWAP: a = 4, b = 5
```

8. 매크로 함수

- SWAP_INT() 매크로 함수를 여러 줄에 정의

코드 6-31 SwapInt2.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   #define SWAP_INT(x, y) \
5       int t = (x);       \
6       (x) = (y);         \
7       (y) = t;
8
9   int main(void)
10  {
11      int a = 5;
12      int b = 4;
13      printf("Before SWAP: a = %d, b = %d\n", a, b);
14      SWAP_INT(a, b);
15      printf("After SWAP: a = %d, b = %d\n", a, b);
16      return 0;
17  }
```

SWAP_INT 매크로를 정의
마지막 줄에는 '\'를 붙이지 않음

<실행 결과>

```
Before SWAP: a = 5, b = 4
After SWAP: a = 4, b = 5
```

8. 매크로 함수

- 기존에 정의된 매크로를 이용해서 새로운 매크로 만들기

코드 6-32 UsePredefinedMacro.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   #define INT1 5
5   #define INT2 (INT1 + 5)
6   #define MUL10(n) ((n) * INT2)
7
8   int main(void)
9   {
10      printf("INT2 = %d\n", INT2);
11      printf("MUL10(5) = %d\n", MUL10(5));
12      return 0;
13  }
```

4줄에서 정의된 INT1을 사용해서 새로운 매크로 상수를 생성

5줄에서 정의된 INT2를 매크로 함수에서 사용

<실행 결과>

```
INT2 = 10
MUL10(5) = 50
```

8. 매크로 함수

- 매크로 상수를 만들고 출력하기

코드 6-33 MacroFunc.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   int main(void)
5   {
6       #define PI 3.141519
7       printf("Pi = %f\n", PI);
8
9       #define MAX(x, y) (((x) > (y)) ? (x) : (y))
10      printf("MAX(3, 5) = %d\n", MAX(3, 5));
11      return 0;
12  }
```

PI를 매크로 상수로 지정
7줄에서 PI를 사용하기 전에만 작성하면 코드 어디에 있어도 됨

x와 y를 받고 둘 중 큰 값을 반환하는 함수 구현. 마찬가지로 함수 사용 전에 정의되면 됨

<실행 결과>

```
Pi = 3.141519
MAX(3, 5) = 5
```

8. 매크로 함수

매크로 함수의 장단점

- 장점
  - 함수보다 빠름
  - 자료형을 구분하지 않음

- 단점
  - 큰 값으로 반환되는 변수는 두 번 증가
  - 따라서 결과가 예상과 다르게 나타남

8. 매크로 함수

매크로 함수의 장단점

단점 예시

- MAX() 함수에 a++와 b++를 전달한다고 가정
- a와 b는 정수 변수이고, 각각 1과 2로 초기화되어 있었다고 가정

```
((a++) > (b++)) ? (a++) : (b++))
```

8. 매크로 함수

매크로 함수의 장단점

단점 예시 코드

코드 6-34 MaxFunc.c

```
1   #define _CRT_SECURE_NO_WARNINGS
2   #include <stdio.h>
3
4   int max(int x, int y)
5   {
6       return (x > y) ? x : y;
7   }
8   #define MAX(x, y) (((x) > (y)) ? (x) : (y))
9
10  int main(void)
11  {
12      int a = 1;
13      int b = 2;
14      int m = max(a++, b++);
15      printf("max(a, b) = %d, a = %d, b = %d\n", m, a, b);
16
17      a = 1;
18      b = 2;
19      m = MAX(a++, b++);
20      printf("MAX(a, b) = %d, a = %d, b = %d\n", m, a, b);
21      return 0;
22  }
```

max() 함수를 호출
함수에 1, 2가 전달되고 각 변수들은 1씩 증가

MAX() 매크로로 치환
매크로 함수 인자에 1, 2가 전달되고 둘 중 작은 변수는 1, 큰 변수는 2가 증가

<실행 결과>

```
max(a, b) = 2, a = 2, b = 3
MAX(a, b) = 3, a = 2, b = 4
```