

0. C 복습

Contents

- ❑ 구조체
- ❑ file 입출력
- ❑ 동적 할당 및 포인터의 사용
- ❑ 재귀 함수
- ❑ 함수 인자 전달 방식
- ❑ 배열을 함수 인자로 전달

구조체

```
struct student{  
    char name[20];  
    int number;  
};
```

* 구조체 변수 초기값

* 구조체 변수 대입

* 구조체 변수 동등비교

- ❑ struct student s1, s2;
- ❑ struct student s[10];

- ❑ struct student s3={"lee", 100};
- ❑ s1의 이름에 "kim", 번호에 200 저장하려면?
 - s1.name = "kim"; ?
 - s1.number=200;
- ❑ s2={"hong", 300}; ?
- ❑ s1=s2; ?

구조체 변수의 초기화

- 변수 선언 시 중괄호를 이용한 초기화 지정이 가능
 - 초기화 값은 중괄호 내부에서 각 멤버 정의 순서대로 초기값을 쉼표로 구분하여 기술
 - 기술되지 않은 멤버값은 자료형에 따라 기본값인 0, 0.0, 'W0' 등으로 저장

```
struct account
{
    char name[12];        //계좌주이름
    int actnum;           //계좌번호
    double balance;       //잔고
};

struct account mine = {"홍길동", 1001, 300000 };
```

그림 13-9 구조체 변수의 초기화

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

void main() {

    struct student {
        char name[20];
        int number;
    };

    struct student s1, s2;
    struct student s[10];
    struct student s3 = { "lee", 100 };

    strcpy(s1.name, "kim");
    s1.number = 200;
    printf("이름: %s, 번호: %d\n", s1.name, s1.number);
    strcpy(s2.name, "hong");
    s2.number = 300;
    printf("이름: %s, 번호: %d\n", s2.name, s2.number);
    s2 = s1;
    printf("이름: %s, 번호: %d\n", s2.name, s2.number);
}
```

```

#include <stdio.h>
#include <string.h>

void printstudent(struct student);

void main() {

    struct student {
        char name[20];
        int number;
    };

    struct student s1, s2;
    struct student s[10];
    struct student s3 = { "lee", 100 };

    strcpy(s1.name, "kim");
    s1.number = 200;
    //printf("이름: %s, 번호: %d\n", s1.name, s1.number);
    strcpy(s2.name, "hong");
    s2.number = 300;
    //printf("이름: %s, 번호: %d\n", s2.name, s2.number);
    s2 = s1;
    //printf("이름: %s, 번호: %d\n", s2.name, s2.number);
}

void printstudent(struct student s) {
    printf("이름: %s, 번호: %d\n", s.name, s.number);
}

```

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

```
struct student {
    char name[20];
    int number;
};
```

```
void printstudent(struct student);
```

```
void main() {

    struct student s1, s2;
    struct student s[10];
    struct student s3 = { "lee", 100 };

    strcpy(s1.name, "kim");
    s1.number = 200;
    printstudent(s1);
    strcpy(s2.name, "hong");
    s2.number = 300;
    printstudent(s2);
    s2 = s1;
    printstudent(s2);
}
```

```
void printstudent(struct student s) {
    printf("이름: %s, 번호: %d\n", s.name, s.number);
}
```

```
struct student{  
    char name[20];  
    int number;  
};
```

```
typedef struct student student;  
student s1, s2;
```



```

#include <stdio.h>
#include <string.h>

struct student {
    char name[20];
    int number;
};

void printstudent(struct student);

void main() {

    struct student s1, s2;
    struct student s[10];
    struct student s3 = { "lee", 100 };

    strcpy(s1.name, "kim");
    s1.number = 200;
    printstudent(s1);
    strcpy(s2.name, "hong");
    s2.number = 300;
    printstudent(s2);
    s2 = s1;
    printstudent(s2);
}

void printstudent(struct student s) {
    printf("이름: %s, 번호: %d\n", s.name, s.number);
}

```

```

#include <stdio.h>
#include <string.h>

struct student {
    char name[20];
    int number;
};
typedef struct student student;

void printstudent(student);

void main() {

    student s1, s2;
    student s[10];
    student s3 = { "lee", 100 };

    strcpy(s1.name, "kim");
    s1.number = 200;
    printstudent(s1);
    strcpy(s2.name, "hong");
    s2.number = 300;
    printstudent(s2);
    s2 = s1;
    printstudent(s2);
}

void printstudent(student s) {
    printf("이름: %s, 번호: %d\n", s.name, s.number);
}

```

텍스트 파일 입력

- text파일로부터 필요한 데이터를 읽어 들여 사용할 경우
 - 데이터를 저장할 변수 VAR 정의(필요 시 구조체);
 - FILE *fp=fopen("a.txt", "r");
 - if(!fp) ... // fp==NULL인 경우 메시지 출력 후 exit(1)
 - char line[100]; //파일로부터 한 줄씩 받아 저장할 임시공간
 - fgets(line, 100, fp);
 - while(!feof(fp)){
 - (1) sscanf 이용하여 line으로부터 분해하여 VAR에 저장
 - (2) (필요시) 그외 해야 하는 일...
 - (3) fgets(line, 100, fp);}
- ※ 경우에 따라 sscanf이외에 fscanf를 쓸 수도 있음.

텍스트 파일 출력

- ❑ 표준입력으로부터 필요한 데이터를 읽어 들여 **text** 파일로 저장할 경우
 - 데이터를 저장할 변수 **VAR** 정의(필요 시 구조체);
 - FILE *fp=fopen("a.txt", "w");
 - if(!fp) ... // fp==NULL인 경우 메시지 출력 후 exit(1)
 - char line[100]; //파일로부터 한 줄씩 받아 저장할 임시공간
 - fgets(line, 100, **stdin**);
 - while(!feof(**stdin**)){ //ctrl-Z 입력받을 때까지
 - (1) fputs(line, fp); //a.txt에 line을 쓴다.
 - (2) 필요시 line으로부터 **VAR**에 정보를 저장하여 작업함.(sscanf)
 - (3) fgets(line, 100, **stdin**);
 - }
- ❑ ※ 경우에 따라 fputs이외에 다른 함수를 쓸 수 있음.

파일 입출력 예제 - 1(1)

- 표준입력으로부터 데이터를 입력 받아 city.txt파일을 만드시오.

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

void main() {
    FILE *fp = fopen("city.txt", "w");
    char line[100];
    if (!fp) {
        puts("Cannot open the file");
        exit(1);
    }
    fgets(line, 100, stdin);
    while(!feof(stdin)){
        fputs(line, fp);
        fgets(line, 100, stdin);
    }
    fclose(fp);
}
```



```
C:\WINDOWS#
Seoul 1000
Busan 350
Daegu 250
Daejeon 153
Incheon 290
Gwangju 148
Jeju 60
Ulsan 116
^Z
```



```
city - 메모장
파일(F)  편집(E)  서식(S)
Seoul 1000
Busan 350
Daegu 250
Daejeon 153
Incheon 290
Gwangju 148
Jeju 60
Ulsan 116
```

1) “city.txt”의 위치?

파일 입출력 예제 - 1(2)

- 문제 1(1)에서 "city.txt"로부터 한 라인씩 입력 받아, 적절한 구조체 배열 원소에 저장.
 - struct city 정의
 - sscanf사용

```
#include <stdio.h>

struct city {
    char name[20];
    int pop;
};

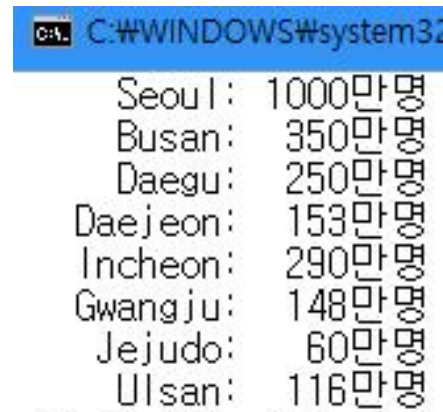
typedef struct city CITY;
```



city - 메모장

파일(F) 편집(E) 서식(S)

Seoul 1000
Busan 350
Daegu 250
Daejeon 153
Incheon 290
Gwangju 148
Jeju 60
Ulsan 116



C:\WINDOWS\system32

Seoul: 1000만명
Busan: 350만명
Daegu: 250만명
Daejeon: 153만명
Incheon: 290만명
Gwangju: 148만명
Jeju: 60만명
Ulsan: 116만명

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

```
struct city {  
    char name[20];  
    int pop;  
};  
typedef struct city CITY;  
void main() {  
    FILE *fp = fopen("city.txt", "r");  
    CITY c[10];  
    char line[100];  
    int i=0, num;  
    if(!fp) {  
        puts("Cannot open the file");  
        exit(1);  
    }  
    fgets(line, 100, fp);  
    while (!feof(fp)) {  
        sscanf(line, "%s %d", c[i].name, &c[i].pop);  
        fgets(line, 100, fp);  
        i++;  
    }  
    fclose(fp);  
    num = i; //도시 개수  
    for (i = 0; i < num; i++) {  
        printf("%10s: %4d만명\n", c[i].name, c[i].pop);  
    }  
}
```

city - 메모장

파일(F) 편집(E) 서식(S) 뷰(V) 도움말(H)

Seoul 1000
Busan 350
Daegu 250
Daejeon 153
Incheon 290
Gwangju 148
Jejudo 60
Ulsan 116

C:\WINDOWS\system32

Seoul: 1000만명
Busan: 350만명
Daegu: 250만명
Daejeon: 153만명
Incheon: 290만명
Gwangju: 148만명
Jejudo: 60만명
Ulsan: 116만명

파일 입출력 예제 - 2(1)

- "in.txt"파일로부터 열 개의 숫자를 int arr[10]에 저장 후, arr[]를 이용하여 "out.txt"파일에 저장하는 프로그램을 작성하시오.

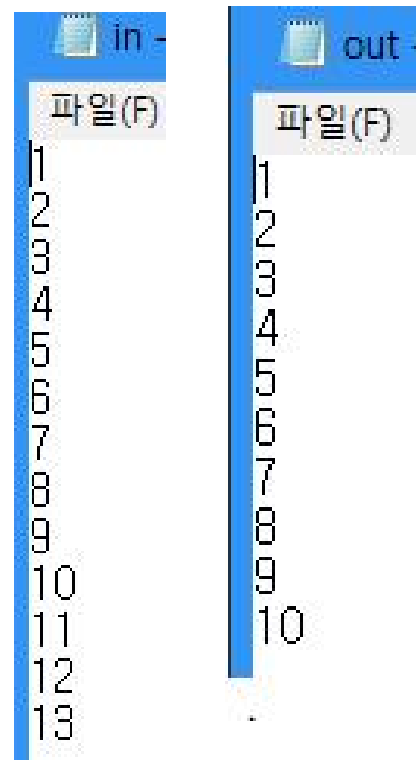
```
#include <stdio.h>

void main() {

    FILE *fp = fopen("in.txt", "r");
    int i, arr[10];
    if (!fp) {
        puts("Cannot open the file");
        exit(1);
    }
    for (i = 0; i < 10; i++) {
        fscanf(fp, "%d", &arr[i]);
    }
    fclose(fp);
    fp = fopen("out.txt", "w");
    if (!fp) {
        puts("Cannot open the file");
        exit(1);
    }
    for (i = 0; i < 10; i++) {
        fprintf(fp, "%d\\n", arr[i]);
    }
    fclose(fp);
}
```

1) "in.txt"의 위치?

2) 파일 내부 숫자의 개수를 모를 경우?



파일 입출력 예제 -2(2)

- 문제 2(1)에서 전체 원소를 out.txt에 저장: feof 사용

```
#include <stdio.h>
#define MAX 50
```

(주의) in.txt에 숫자 입력 시에 13에서 줄바꿈시 유의. (줄바꿈 하지말것)

```
void main() {
    FILE *fp1 = fopen("in.txt", "r"), *fp2=fopen("out.txt", "w");
    int i=0, arr[MAX];
    if (!fp1 || !fp2) {
        puts("Cannot open the file");
        exit(1);
    }
    while (!feof(fp1)) {
        fscanf(fp1, "%d", &arr[i]);
        fprintf(fp2, "%d\\n", arr[i]);
        i++;
    }
    fclose(fp1);
    fclose(fp2);
}
```

in -	out -
파일(F)	파일(F)
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13

동적할당 및 포인터의 사용

```
int *pi;  
*pi = 3;  
➔ error!
```

```
int *pi;  
int i=3;  
pi=&i;
```

```
int *pi;  
pi=(int*)malloc(sizeof(int));  
*pi=100;
```

pi가 가리키는 곳을 '인위적으로' 만들자! ➔ malloc 함수 이용.
pi=(int *)malloc(sizeof(int));

동적할당 및 포인터의 사용

- 동적 메모리 할당 함수
 - 힙(heap)에 할당

이부분이 없으면 에러! (저장할 공간없음)

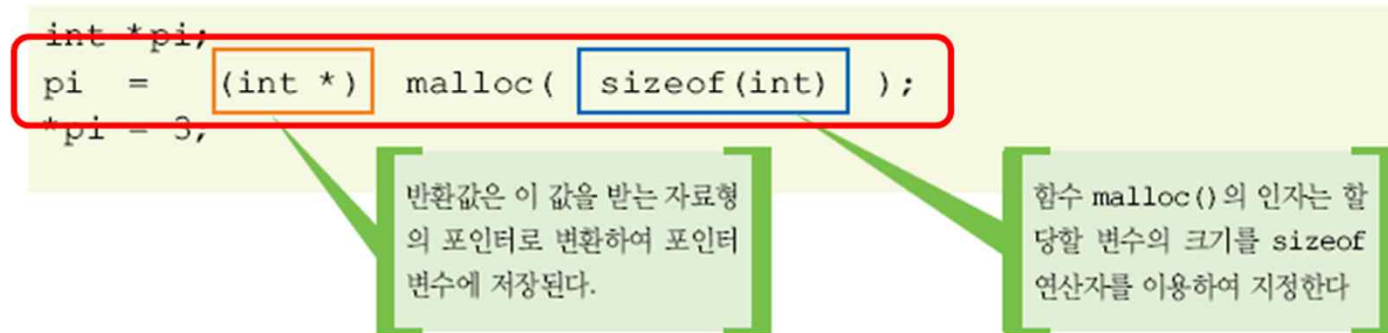


그림 18.1 함수 malloc()으로 정수형 저장공간 할당

```
/* memoryalloc.c */
```

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

```
int main(void) {
    int *pi, *ary, i = 0;
    double *pd;
```

```
    printf("문장 : pi = (int *) malloc( sizeof(int) );\n");
    pi = (int *) malloc( sizeof(int) );
    *pi = 3;
    printf("*pi = %d\n\n", *pi);
    free(pi);
```

```
    printf("문장 : pd = (double *) malloc( sizeof(double) );\n");
    pd = (double *) malloc( sizeof(double) );
    *pd = 3.15;
    printf("*pd = %5.2lf\n\n", *pd);
    free(pd);
```

```
    printf("문장 : ary = (int *) malloc( sizeof(int)*3 );\n");
    ary = (int *) malloc( sizeof(int)*3 );
    ary[0] = 10; ary[1] = 11; ary[2] = 12;
    for (i = 0; i < 3; i++)
        printf("ary[%d] = %d ", i, *(ary + i));
    printf("\n");
    free(ary);
```

```
    return 0;
```

```
}
```

C:\Windows\system32\cmd.exe

```
문장 : pi = (int *) malloc( sizeof(int) );
*pi = 3
```

```
문장 : pd = (double *) malloc( sizeof(double) );
*pd = 3.15
```

```
문장 : ary = (int *) malloc( sizeof(int)*3 );
ary[0] = 10 ary[1] = 11 ary[2] = 12
```

pi int 형 4바이트의 저장공간



그림 18.1 함수 malloc()으로 정수형 저장공간 할당

ary

int 형 4바이트가 3개인 저장공간

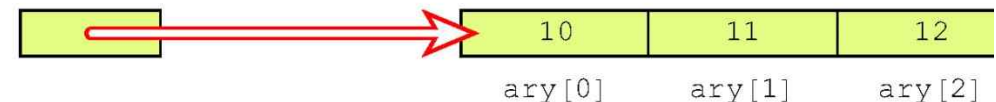


그림 18.3 함수 malloc()으로 정수형 여러 저장공간 할당

```

#include <stdio.h>
#include <stdlib.h>
#define NUM 10

```

```

int main()
{

```

```

    int i, j;
    int *p, *p_arr;

```

```

    p_arr = (int *)malloc(sizeof(int) * NUM);
    for (i = 0; i < NUM; i++)
        p_arr[i] = i;

    for (i = NUM - 1; i >= 0; i--)
        printf("%d ", p_arr[i]);

```

```

    printf("\n\n");

```

```

    p = p_arr; i = 0;
    while (i < NUM) {
        printf("p: %d    *p: %d\n", p, (*p));
        p++; i++;
    }

```

```

    puts("");

```

```

    p = &p_arr[NUM - 1];
    while ((*p) >= 0) {
        printf("p: %d    *p: %d\n", p, (*p));
        p--;
    }

```

```

}

```

9 8 7 6 5 4 3 2 1 0

p: 46060104 *p: 0
p: 46060108 *p: 1
p: 46060112 *p: 2
p: 46060116 *p: 3
p: 46060120 *p: 4
p: 46060124 *p: 5
p: 46060128 *p: 6
p: 46060132 *p: 7
p: 46060136 *p: 8
p: 46060140 *p: 9

p: 46060140 *p: 9
p: 46060136 *p: 8
p: 46060132 *p: 7
p: 46060128 *p: 6
p: 46060124 *p: 5
p: 46060120 *p: 4
p: 46060116 *p: 3
p: 46060112 *p: 2
p: 46060108 *p: 1
p: 46060104 *p: 0

재귀함수(Recursive function)

$$\begin{aligned} \square \quad & f(n) = f(n-1) + 2 \quad (n > 0) \\ & = 0 \quad (n = 0) \end{aligned}$$

$f(4)$ 의 값은?

$$\begin{aligned} f(4) &= \underline{f(3)} + 2 \\ &= (\underline{f(2)} + 2) + 2 \\ &= ((\underline{f(1)} + 2) + 2) + 2 \\ &= \\ &= (((\underline{f(0)} + 2) + 2) + 2) + 2 \\ &= \underline{0} + 2 + 2 + 2 + 2 \end{aligned}$$

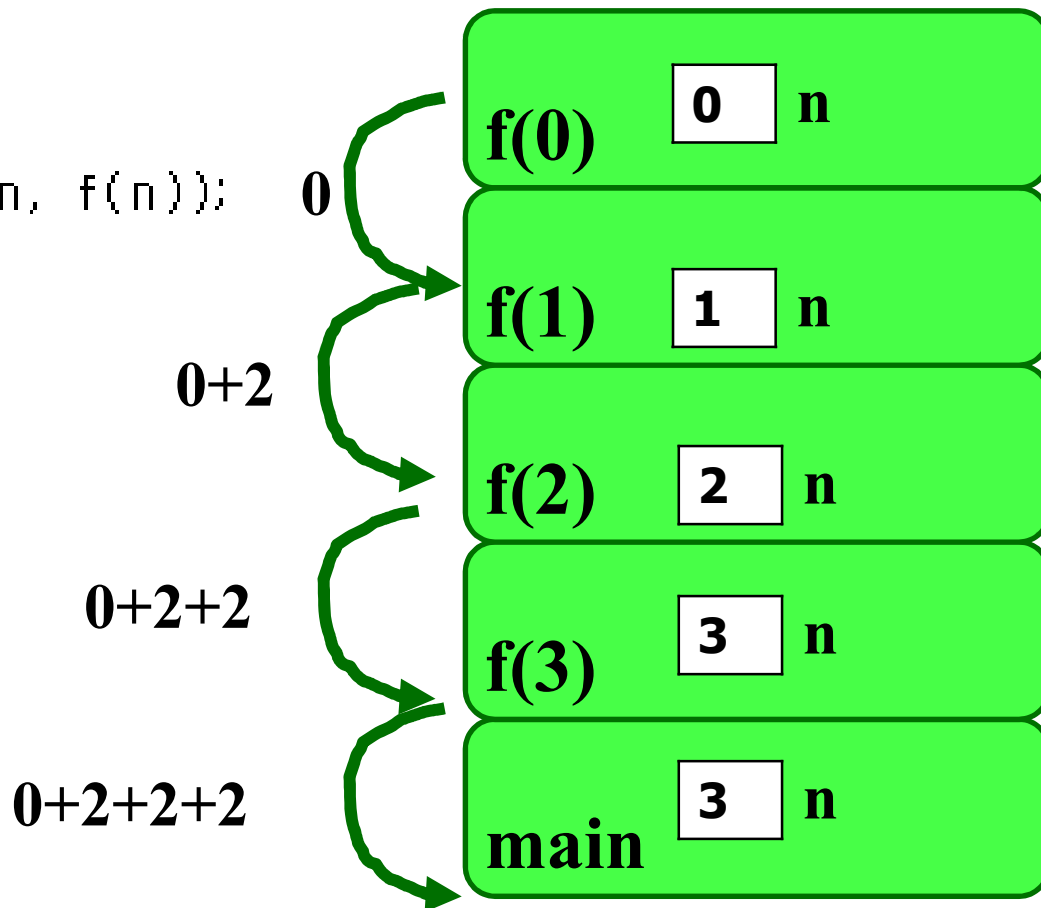
$$\begin{aligned} \square \quad & f(n) = f(n-1) + 2 \quad (n > 0) \\ & = 0 \quad (n = 0) \end{aligned}$$

$$\begin{aligned} f(3) &= f(2) + 2 \\ &= (f(1) + 2) + 2 \\ &= ((f(0) + 2) + 2) + 2 \\ &= (((0) + 2) + 2) + 2 \end{aligned}$$

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

```
int f(int);
void main() {
    int n = 3;
    printf("f(%d)=%d\n", n, f(n));
}
```

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



재귀 특성

- 재귀 함수(recursive function)
 - 함수구현에서 자신 함수를 호출하는 함수
 - 재귀적 특성을 표현하는 알고리즘
 - 재귀 함수를 이용하면 문제를 쉽게 해결할 수 있고 이해하기도 쉬움

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

그림 10-18 n!의 정의와 재귀 특성

재귀 특성

□ 수학 수식에서 재귀적 특성

- n 계승(n factorial)을 나타내는 수식 $n!$
 - $1 * 2 * 3 * \dots * (n-2) * (n-1) * n$ 을 의미
 - 즉 $n!$ 의 정의에서 보듯 계승은 재귀적 특성
- $n!$ 을 구하기 위해서 $(n-1)!$ 을 먼저 구한다면 쉽게 $n!$ 을 구할 수 있음
 - $(n-1)!$ 을 구하기 위해서는 다시 $(n-2)!$ 이 필요
- $n!$ 을 함수 $\text{factorial}(n)$ 로 구현한다면
 - 함수 $\text{factorial}(n)$ 구현에서 다시 $\text{factorial}(n-1)$ 을 호출하여 그 결과를 이용 가능

재귀 특성

```
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));
}
```

그림 10-19 n!을 위한 함수 factorial()

재귀 함수

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

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

```
// file: factorial.c
#include <stdio.h>

int factorial(int); //함수원형

int main(void)
{
    for (int i = 1; i <= 10; i++)
        printf("%2d! = %d\n", i, factorial(i));

    return 0;
}

// n! 구하는 재귀함수
int factorial(int number)
{
    if (number <= 1)
        return 1;
    else
        return (number * factorial(number - 1));
}
```

5!

```
// n! 구하는 재귀함수
int factorial(int number)
{
    if (number <= 1)
        return 1;
    else
        return (number * factorial(number - 1));
}
```

factorial(5)

return 5*factorial(4)

return 4*factorial(3)

return 3*factorial(2)

return 2*factorial(1)

return 1

factorial(1) number=1

factorial(2) number=2

factorial(3) number=3

factorial(4) number=4

factorial(5) number=5

main

재귀 함수의 실행

- 함수 `factorial(n)`에서 `factorial(3)`을 호출한 경우 실행과정
 - `factorial(3)`을 호출하면 함수 `factorial(3)` 내부에서 다시 `factorial(2)`를 호출
 - `factorial(2)`에서는 다시 `factorial(1)`을 호출
 - 결국 `factorial(1)` 내부에서 `return 1`을 실행
 - 다시 `factorial(2)`로 돌아와 반환값 1을 이용하여 `return (2*1)`을 실행
 - 계속해서 `factorial(3)`으로 돌아와 `factorial(2)`의 결과값인 2를 이용
 - `return (3*2)`를 실행하면 결국 6을 반환
- 재귀 함수 장단점
 - 일반적으로 재귀 함수는 함수의 호출이 계속되면 시간도 오래 걸리고 메모리의 사용도 많다는 단점

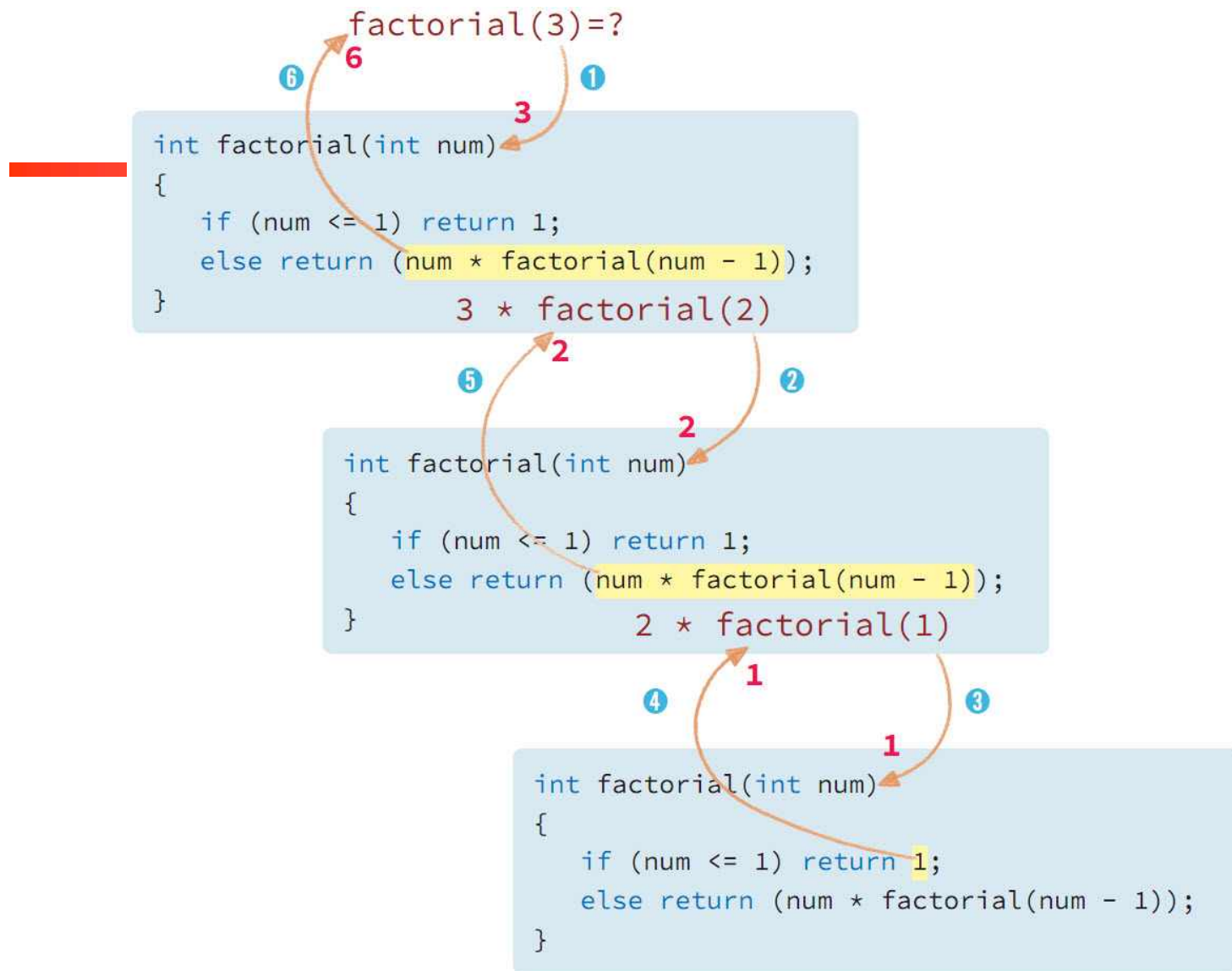


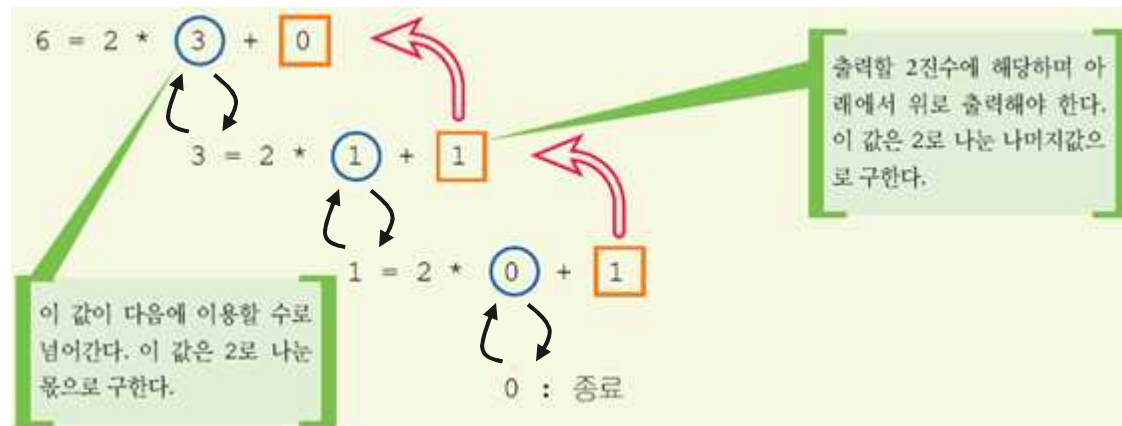
그림 10-20 재귀함수의 실행

2진수 구하기

- ❑ 십진수를 이진수로 바꾸기
 - 재귀적 특성을 가짐
- ❑ 35를 2진수로...

2진수 구하기

- 십진수를 이진수로 바꾸기
 - 재귀적 특성을 가짐



```
n을 2진수로 표현하려면 :  
if (n>0)  
{  
    (n/2)을 2진수 표현하고  
    (n%2) 결과를 출력  
}  
else  
{  
    종료  
}
```



```
void binary(int number)  
{  
    if (number > 0)  
    {  
        binary(number/2);  
        printf("%d", number % 2);  
    }  
}
```

```

#include <stdio.h>

void binary(int number); //함수 원형

int main(void)
{
    int decimal;

    printf("정수 0을 입력하면 프로그램이 종료합니다.\n");
    printf("양의 정수를 하나 입력하세요. >> ");
    while ((scanf("%d", &decimal) && decimal > 0))
    {
        printf("양의 정수 %7d의 2진수는 >> ", decimal);

        //재귀함수 호출
        binary(decimal);
        printf("\n\n양의 정수를 하나 입력하세요. >> ");
    }

    return 0;
}

```

```

// 이진수를 구하는 재귀 함수
void binary(int number)
{
    int bin;

    if (number > 0)
    {
        //변수 bin에 나머지를 저장
        bin = number % 2;
        number /= 2;

        //재귀 호출
        binary(number);
        printf("%d", bin);
    }

    return;
}

```

C:\Windows\system32\cmd.exe

```

정수 0을 입력하면 프로그램이 종료합니다.
양의 정수를 하나 입력하세요. >> 35
양의 정수      35의 2진수는 >> 100011

양의 정수를 하나 입력하세요. >> 3334
양의 정수     3334의 2진수는 >> 110100000110

양의 정수를 하나 입력하세요. >> 0
계속하려면 아무 키나 누르십시오 . . .

```


binary(35) -{bin=1;number=17;binary(17); print “1”}

- {bin=1;number=8;binary(8); print “1”}

// 이진수를 구하는 재귀 함수

void binary(int number)

{

int bin;

if (number > 0)

{

//변수 bin에 나머지를 저장

bin = number % 2;
number /= 2;

//재귀 호출

binary(number);
printf("%d", bin);

}

return;

}

-{bin=0;number=4;binary(4); print “0”}

-{bin=0;number=2;binary(2); print “0”}

-{bin=0;number=1;binary(1); print “0”}

- {bin=1;number=0;binary(0);print “1”}

- {return;}

35=100011₍₂₎

다음의 결과는?

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

1 계속하려면 아무 키나 누르십시오 . . .

```
void printarr(int[]);
```

```
void main() {  
    int arr[] = { 1, 2, 3, 4, 5, 6 };  
    printarr(arr);  
}
```

```
void printarr(int arr[]) {  
    int size = sizeof(arr) / sizeof(int);  
    for (int i = 0; i < size; i++) {  
        printf("%d,", arr[i]);  
    }  
}
```

배열의 크기를 인자로 전달

```
#include <stdio.h>

void printarr(int*, int);

void main() {

    int arr[] = { 1, 2, 3, 4, 5, 6 };
    int size = sizeof(arr) / sizeof(int);
    printarr(arr, size);
}

void printarr(int arr[], int size) {

    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
}
```

같은 의미로 모두 사용할 수 있다

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

```
int sumaryf(int *ary, int SIZE)
{
    ...
}
```

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

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

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

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

그림 14-6 함수헤더의 배열 인자와 함수정의에서 다양한 배열원소의 참조방법

함수 인자전달방식

- Value (값)에 의한 호출
- Reference (참조)에 의한 호출

- 실인자/형식인자

```
int add(int a, int b){  
    return a+b;  
}
```

..

```
int c=add(3, 4)
```

함수 호출에서 전달의 차이

- call by value
 - 함수의 인자로 일반 변수를 사용하는 방식

```
int main(void)
{
    int number = 10;
    increment(number);

    return 0;
}

void increment(int num)
{
    num++;
}
```

number: 10
(main함수)

num: 10 → 11
(increment함수)

- 실인자로 사용하는 변수 number에는 영향이 없음

함수 호출에서 전달의 차이

- ❑ call by reference
 - 함수의 인자로 포인터 변수를 사용

```
int main(void)
{
    int number = 10;
    incrementbyaddress(&number);

    return 0;
}

void incrementbyaddress(int *num)
{
    (*num)++; // ++(*num)
}
```

number: 10 → 11

(main)

num:

(incrementbyaddress)

- 실인자로 사용하는 변수 number 값이 1 증가함