

C언어 강의자료

문정욱

A decorative graphic consisting of several light blue squares of varying sizes arranged in a stepped pattern on the left side of the slide.

C언어 맛 보기 4

지역변수와 전역변수

- 내부영역(internal area)
 - 함수의 내부영역
- 외부영역(external area)
 - 함수 외부영역

<pre>#include <stdio.h> int a; double b=2.6;</pre>	external area
<pre>int funct(void) { double d=1.5; int e=4, f; printf("%f\n", b); return e; }</pre>	internal area
<pre>int g=2;</pre>	external area
<pre>int main(void) { double h=1.44; int i; i = funct(); printf("%d\n", i); printf("%d\n", a + g); return 0; }</pre>	internal area
<pre>int j; double k=2.6;</pre>	external area

지역변수와 전역변수

- 내부영역(internal area)
 - 지역변수(local variable) 선언
- 외부영역(external area)
 - 전역변수(global variable) 선언

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

```
int a;  
double b=2.6;
```

global variable

```
int funct(void)  
{
```

```
    double d=1.5;  
    int e=4, f;
```

local variable

```
    printf("%f\n", b);  
    return e;
```

```
}
```

```
int g=2;
```

global variable

```
int main(void)  
{
```

```
    double h=1.44;  
    int i;
```

local variable

```
    i = funct();  
    printf("%d\n", i);  
    printf("%d\n", a + g);  
    return 0;
```

```
}
```

```
int j;  
double k=2.6;
```

global variable

지역변수와 전역변수

- 내부영역(internal area)
 - 지역변수(local variable) 선언
 - 문장(statement) 서술 가능
- 외부영역(external area)
 - 전역변수(global variable) 선언
 - **문장(statement) 서술 불가능**

```
#include <stdio.h>

int a;
double b=2.6;

int funct(void)
{
    double d=1.5;
    int e=4, f;

    return e;
}

printf("%f\n", b);
int g=2;

int main(void)
{
    double h=1.44;
    int i;

    i = funct();
    printf("%d\n", i);
    return 0;
}

printf("%d\n", a + g);
int j;
double k=2.6;
```

external area

external area

syntax error

external area

지역변수와 전역변수

- 지역변수의 참조 범위
 - 선언된 위치에서부터
해당 블록(block)의 끝까지
참조 가능

```
#include <stdio.h>

int a;
double b=2.6;

int funct(void)
{
    double d=1.5;
    int e=4, f;
    printf("%f\n", d);
    return e;
}

int g=2;

int main(void)
{
    double h=1.44;
    int i;
    i = funct();
    printf("%d\n", i);
    printf("%d\n", a + i);
    return 0;
}

int j;
double k=2.6;
```

지역변수와 전역변수

- 전역변수의 참조 범위
 - 선언된 위치에서부터
파일 끝까지 참조 가능

```
#include <stdio.h>

int a;
double b=2.6;

int funct(void)
{
    double d=1.5;
    int e=4, f;

    printf("%f\n", b);
    return e;
}

int g=2;

int main(void)
{
    double h=1.44;
    int i;

    i = funct();
    printf("%d\n", i);
    printf("%d\n", a + g);
    return 0;
}

int j;
double k=2.6;
```

The diagram illustrates the scope of global variables. A red arrow labeled "accessible" points from the declaration of `double b=2.6;` to its use in `printf("%f\n", b);` inside the `funct` function. Another red arrow labeled "accessible" points from the declaration of `int g=2;` to its use in `printf("%d\n", a + g);` inside the `main` function. This demonstrates that global variables are accessible from their point of declaration until the end of the file.

지역변수의 참조 범위

- 중첩 블록(nested block)
 - 중첩 블록 안에 선언된 변수는 해당 블록이 끝나는 범위까지 참조 가능하다.

```
#include <stdio.h>

int main(void)
{
    int a;
    int b = 3;

    a = 3;
    a = a + 1;
    b = a * b;
    {
        int c;

        c = a - b;
        b = c * 3;
    }
    return 0;
}
```


지역 변수

- 함수 입력 인자
 - 함수의 입력 인자는 함수의 지역변수이다.
 - 즉, 함수 안에서만 쓰인다.

```
#include <stdio.h>
int sum(int n)
{
    int i, s=0;
    for(i=1;i<=n;++i)
        s+=i;
    return s;
}

int main(void)
{
    printf("sum == %d\n", sum(10));
    return 0;
}
```

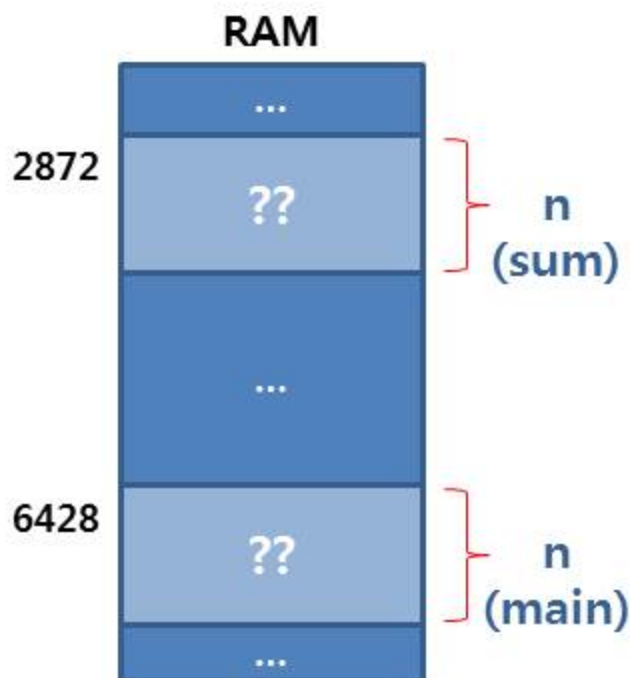
sum함수의 지역변수

입출력 결과

```
sum == 55
계속하려면 아무 키나 누르십시오 . . .
```

지역 변수

- 같은 이름의 변수
 - 서로 다른 함수에서 같은 이름의 변수는 **서로 다른 변수**이다.
 - **개별 메모리 공간**을 갖는다.



```
#include <stdio.h>
int sum(int n)
{
    int i, s=0;
    for(i=1;i<=n;++i)
        s+=i;
    return s;
}
int main(void)
{
    int n;
    scanf("%d", &n);
    printf("sum == %d\n", sum(n));
    return 0;
}
```

Annotations:

- Red arrow pointing to `#include <stdio.h>`: **sum함수의 지역변수**
- Red arrow pointing to `int n` in the `sum` function: **이름만 같을 뿐 서로 다른 변수**
- Red arrow pointing to `int n;` in the `main` function: **main함수의 지역변수**

지역 변수

호출부의 변수와 이름이 다른 경우

```
#include <stdio.h>

int sum(int b)
{
    int i, s=0;
    for(i=1; i<=b; ++i)
        s+=i;
    return s;
}

int main(void)
{
    int n;

    scanf("%d", &n);
    printf("sum == %d\n", sum(n));
    return 0;
}
```

일반적으로
정의부 변수의 이름은
호출부 변수와 다르게 함.

호출부의 변수와 이름이 같은 경우

```
#include <stdio.h>

int sum(int n)
{
    int i, s=0;
    for(i=1; i<=n; ++i)
        s+=i;
    return s;
}

int main(void)
{
    int n;

    scanf("%d", &n);
    printf("sum == %d\n", sum(n));
    return 0;
}
```

호출부 변수의
이름과 같아도 됨.
하지만,
서로 다른 변수이다.

변수 충돌(variable collision)

- 지역변수와 외부지역변수의 충돌(variable collision)
 - 외부지역변수와 이름이 같은 내부지역변수가 블록 안에 있으면 **내부지역변수를 참조**한다. (inner access)
 - 외부지역변수와 이름이 같은 내부지역변수가 블록 안에 없으면 **외부지역변수를 참조**한다. (outer access)

입출력 결과

```
1
2
1
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

int main(void)
{
    int a=1;
    printf("%d\n", a);
    {
        int a=2;
        printf("%d\n", a);
    }
    printf("%d\n", a);
    return 0;
}
```


변수 충돌(variable collision)

- 지역변수와 전역변수의 충돌(variable collision)
 - 전역변수와 이름이 같은 지역변수가 블록 안에 있으면 **지역변수를 참조한다. (inner access)**
 - 전역변수와 이름이 같은 지역변수가 블록 안에 없으면 **전역변수를 참조한다. (outer access)**

입출력 결과

2

1

계속하려면 아무 키나 누르십시오 . . .

```
#include <stdio.h>

int a=1; ← outer access

int f(void)
{
    int a=2; ← inner access
    printf("%d\n", a);
    return 0;
}

int g(void)
{
    printf("%d\n", a);
    return 0;
}

int main(void)
{
    f();
    g();
    return 0;
}
```

증감 연산자

전치 연산과 후치 연산의 차이

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

```
int main(void)
```

```
{  
    int a=3, b=3, c=3, d=3;
```

```
    printf("%d\n", ++a);
```

```
    printf("%d\n", b++);
```

```
    printf("%d\n", --c);
```

```
    printf("%d\n", d--);
```

```
    return 0;
```

```
}
```

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

```
int main(void)
```

```
{  
    int a=3, b=3, c=3, d=3;
```

```
    a=a+1; // a==4  
    printf("%d\n", a); // a==4
```

```
    printf("%d\n", b); // b==3  
    b=b+1; // b==4
```

```
    c=c-1; // c==2  
    printf("%d\n", c); // c==2
```

```
    printf("%d\n", d); // d==3  
    d=d-1; // d==2  
    return 0;
```

```
}
```

증감 연산자

- 증감 연산 전/후의 변수의 값 비교

입출력 결과

```
4 4
3 4
2 2
3 2
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

int main(void)
{
    int a=3, b=3, c=3, d=3;

    printf("%d ", ++a );
    printf("%d\n", a);

    printf("%d ", b++ );
    printf("%d\n", b);

    printf("%d ", --c );
    printf("%d\n", c);

    printf("%d ", d-- );
    printf("%d\n", d);
    return 0;
}
```


증감 연산자

전치 증감 연산자의 사용

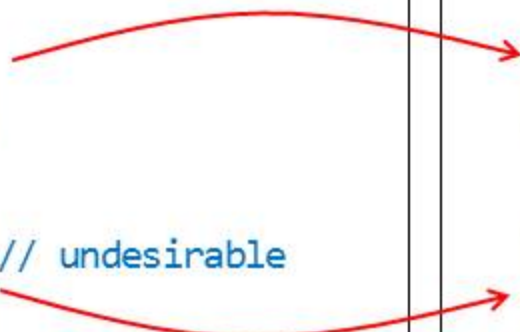
```
#include <stdio.h>

int main(void)
{
    int a,b;

    a=3;
    b=++a;

    a=3;
    b=++a + ++a + ++a; // undesirable

    return 0;
}
```



전치 증감 연산자의 의미

```
#include <stdio.h>

int main(void)
{
    int a,b;

    a=3;

    a=3;

    return 0;
}
```

증감 연산자

후치 증감 연산자의 사용

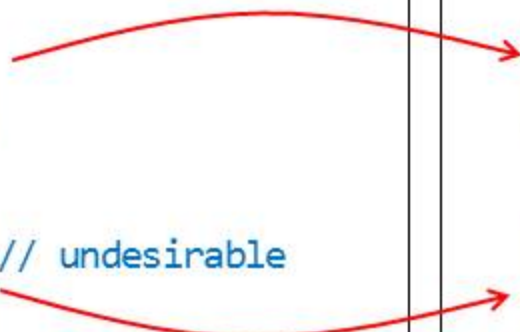
```
#include <stdio.h>

int main(void)
{
    int a,b;

    a=3;
    b=a++;

    a=3;
    b=a++ + a++ + a++; // undesirable

    return 0;
}
```



후치 증감 연산자의 의미

```
#include <stdio.h>

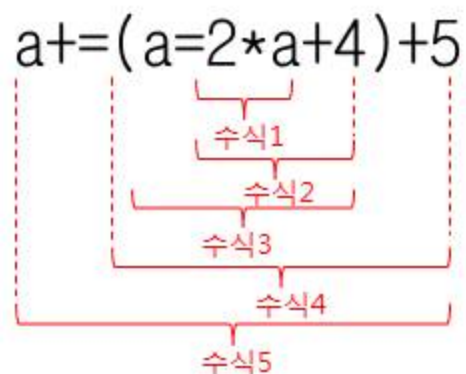
int main(void)
{
    int a,b;

    a=3;

    a=3;
```

수식(expression)

■ 연산 순서



```
#include <stdio.h>

int main(void)
{
    int a;

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

수식(expression)

```
#include <stdio.h>

int main(void)
{
    int a;

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

```
#include <stdio.h>

int main(void)
{
    int a;

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

함수의 형태

■ 인자 전달 (배열)

• 정의부

- 호출부 배열 선언과 동일하게 선언
- 변수의 이름은 같아도 되고 달라도 됨.

• 호출부

- 호출부에서는 배열의 이름만 사용하여 함수를 호출함.

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

```
void print(int a[3])
```

a도 int a[3]이라고 선언한다.

```
{
```

```
    int i;
```

```
    for(i=0;i<3;++i)
```

```
        printf("%d ", a[i]);
```

```
    printf("\n");
```

```
}
```

```
int main(void)
```

```
{
```

```
    int x[3]={11,22,33};
```

```
    print(x);
```

```
    return 0;
```

```
}
```

x를 int x[3]이라고 선언했으므로

함수의 형태

- 인자 전달 (배열)
 - 함수 정의부에서 **배열의 길이는 생략 가능하다.**
※ 배열의 길이 = 배열 원소의 개수

```
#include <stdio.h>

void print(int a[])
{
    int i;

    for(i=0;i<3;++i)
        printf("%d ", a[i]);
    printf("\n");
}

int main(void)
{
    int x[3]={11,22,33};

    print(x);
    return 0;
}
```

배열 길이 생략 가능

함수의 형태

호출부의 배열과 이름이 다른 경우

```
#include <stdio.h>

void print(int a[3])
{
    int i;

    for(i=0;i<3;++i)
        printf("%d ", a[i]);
    printf("\n");
}

int main(void)
{
    int x[3]={11,22,33};

    print(x);
    return 0;
}
```

일반적으로
호출부 배열의 이름과
다르게 함.

호출부의 배열과 이름이 같은 경우

```
#include <stdio.h>

void print(int x[3])
{
    int i;

    for(i=0;i<3;++i)
        printf("%d ", x[i]);
    printf("\n");
}

int main(void)
{
    int x[3]={11,22,33};

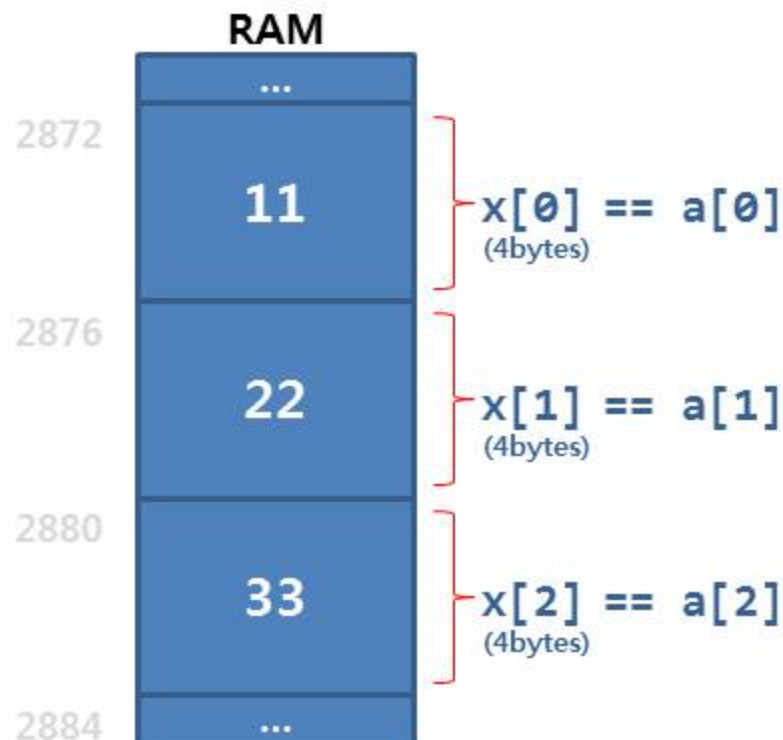
    print(x);
    return 0;
}
```

호출부 배열의
이름과 같아도 됨.

함수의 형태

■ 인자 전달 (배열)

- 배열의 인자 전달
 - 메모리 공간을 전달.
 - 그래서 메모리 공간을 공유하게 됨.



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

```
void print(int a[3]) x의 메모리 공간을 공유함
{
    int i;

    for(i=0; i<3; ++i)
        printf("%d ", a[i]);
    printf("\n");
}
```

```
int main(void)
{
    int x[3]={11,22,33};

    print(x);
    return 0;
}
```

단일변수는 값을 전달 → 메모리 공유 X
배열은 메모리를 전달 → 메모리 공유 O

함수의 형태

일반 변수의 전달

```
#include <stdio.h>

void input(int x0, int x1)
{
    scanf("%d%d", &x0, &x1);
}

int main(void)
{
    int a0=0, a1=1;

    input(a0, a1);
    printf("a0==%d, ", a0);
    printf("a1==%d\n", a1);
    return 0;
}
```

일반 변수는 메모리를 공유하지 않으므로 input 함수에서 x0, x1의 값을 변경하면 main 함수의 a0, a1의 값이 변경되지 않는다.

3 4
a0==0, a1==1

계속하려면 아무 키나 누르십시오 . . .

배열의 전달

```
#include <stdio.h>

void input(int x[2])
{
    scanf("%d%d", &x[0], &x[1]);
}

int main(void)
{
    int a[2]={0,1};

    input(a);
    printf("a[0]==%d, ", a[0]);
    printf("a[1]==%d\n", a[1]);
    return 0;
}
```

배열은 메모리를 공유하므로 input 함수에서 배열 x의 값을 변경하면 main 함수의 배열 a의 값이 변경된다.

3 4
a[0]==3, a[1]==4

계속하려면 아무 키나 누르십시오 . . .

함수의 형태

■ 인자 전달 (배열)

- 첫 번째 print 함수 호출
 - a는 배열 x의 메모리를 공유
- 두 번째 print 함수 호출
 - a는 배열 y의 메모리를 공유

입출력 결과

```
11 22 33
44 55 66
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

void print(int a[3]) a → memory of x
{
    int i;
    for(i=0; i<3; ++i)
        printf("%d ", a[i]);
    printf("\n");
}

int main(void)
{
    int x[3]={11,22,33};
    int y[3]={44,55,66};

    print(x);
    print(y);
    return 0;
}
```

함수의 형태

- 함수 인자 전달 (단일 변수)
 - nine 함수를 호출할 때 x의 값이 변수 a에 전달된다.
 - a는 x는 서로 다른 메모리 공간을 차지한다.
 - 그러므로 a의 값이 바뀌어도 **x의 값은 바뀌지 않는다.**

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

```
void nine(int a)  
{  
    a=9;  
}
```

```
int main(void)  
{  
    int x=1;  
  
    nine(x);  
    printf("x == %d\n", x);  
    return 0;  
}
```

함수 호출 후
x의 값은 변경 안됨.

입출력 결과

x == 1

계속하려면 아무 키나 누르십시오 . . .

함수의 형태

- 함수 인자 전달 (단일 변수)
 - nine 함수를 호출할 때 변수 x(in main)의 값이 변수 x(in nine)에 전달된다.
 - x(in main)는 x(in nine)와 **서로 다른 메모리 공간**을 차지한다.
 - 그러므로 x(in nine)의 값이 바뀌어도 **x(main)의 값은 바뀌지 않는다.**

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

```
void nine(int x)  
{  
    x=9;  
}
```

```
int main(void)  
{  
    int x=1;  
  
    nine(x);  
    printf("x == %d\n", x);  
    return 0;  
}
```

함수 호출 후
x의 값은 변경 안됨.

입출력 결과

x == 1

계속하려면 아무 키나 누르십시오 . . .

함수의 형태

■ 함수 인자 전달 (배열)

- nine 함수를 호출할 때 x의 메모리 공간이 a에 전달된다.
- a는 x의 메모리 공간을 가리킨다.
- a의 요소 값을 바꾸면 **x의 요소 값도 바뀐다.**

※ 인자 전달 후 변수 값의 변화에 대한 원리는 포인터(pointer)를 배운 후 자세히 다룬다.

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

```
void nine(int a[2])  
{  
    a[0]=9;  
    a[1]=99;  
}
```

```
int main(void)  
{  
    int x[2]={1,11};  
  
    nine(x);  
    printf("x == %d, %d\n", x[0], x[1]);  
    return 0;  
}
```

함수 호출 후
x의 값이 변경됨.

입출력 결과

x == 9, 99

계속하려면 아무 키나 누르십시오 . . .

함수의 형태

■ 함수의 반환문

- 함수의 반환 타입이 void가 아닐 경우 함수는 반드시 return문을 실행하고 끝나야 한다.
- 만일 **반환 값 없이 함수가 종료**되면 **논리오류**가 발생할 수 있다.

Logical Error

입출력 결과

```
1
-1
-858993460
```

계속하려면 아무 키나 누르십시오 . . .

```
#include <stdio.h>

int cmp(int n)
{
    if(n>0)
        return 1;
    if(n<0)
        return -1;
    // no return, if n==0.
}

int main(void)
{
    printf("%d\n", cmp(3) );
    printf("%d\n", cmp(-7) );
    printf("%d\n", cmp(0) ); // oops!
    return 0;
}
```


함수의 형태

반환값 반환 방법 1 (바람직하지 못한 방법)

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

```
int cmp(int n)
{
```

```
    if(n>0)
```

```
        return 1;
```

```
    if(n<0)
```

```
        return -1;
```

```
    if(n==0)
```

```
        return 0;
```

```
}
```

```
int main(void)
{
```

```
    printf("%d\n", cmp(3) );
```

```
    printf("%d\n", cmp(-7) );
```

```
    printf("%d\n", cmp(0) );
```

```
    return 0;
```

```
}
```

논리식의 의미를
파악하면
n이 어떤 값이든
return 문이
반드시 실행됨.

하지만, 프로그램 구조만 봤을 때,
return 반드시 실행되는지
확신하기 힘들다.

반환값 반환 방법 2 (바람직한 방법)

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

```
int cmp(int n)
{
```

```
    if(n>0)
```

```
        return 1;
```

```
    if(n<0)
```

```
        return -1;
```

```
    return 0;
```

```
}
```

```
int main(void)
{
```

```
    printf("%d\n", cmp(3) );
```

```
    printf("%d\n", cmp(-7) );
```

```
    printf("%d\n", cmp(0) );
```

```
    return 0;
```

```
}
```

무조건 실행되는
return 문이 있다.

반환값을 반환하지 않는
실수를 줄일 수 있다.

재귀 호출(recursive call)

■ factorial

$$f(n) = n!$$

$$f(n) = \begin{cases} 1 & (n=0) \\ n \cdot f(n-1) & (n \geq 1) \end{cases}$$

입출력 결과

120

계속하려면 아무 키나 누르십시오 . . .

```
#include <stdio.h>

int fact(int n)
{
    if( n==0 ) return 1;
    return n * fact( n-1 );
}

int main(void)
{
    printf("%d\n", fact(5) );
    return 0;
}
```

재귀 호출(recursive call)

■ Fibonacci Sequence

$$f(n) = \begin{cases} 0 & (n=0) \\ 1 & (n=1) \\ f(n-1) + f(n-2) & (n \geq 2) \end{cases}$$

입출력 결과

5
계속하려면 아무 키나 누르십시오 . . .

```
#include <stdio.h>

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

int main(void)
{
    printf("%d\n", fibo(5) );
    return 0;
}
```

재귀 호출 분석

Factorial

```
#include <stdio.h>

int fact(int n)
{
    if( n==0 ) return 1;
    return n * fact( n-1 );
}

int main(void)
{
    printf("%d\n", fact(5) );
    return 0;
}
```

Call Sequence

fact(5) = 120

fact(4) = 24

fact(3) = 6

fact(2) = 2

fact(1) = 1

fact(0) = 1

재귀 호출 분석

Fibonacci

```
#include <stdio.h>

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

int main(void)
{
    printf("%d\n", fibo(4) );
    return 0;
}
```

Call Sequence

$$\text{fibo}(4) = 2 + 1 = 3$$

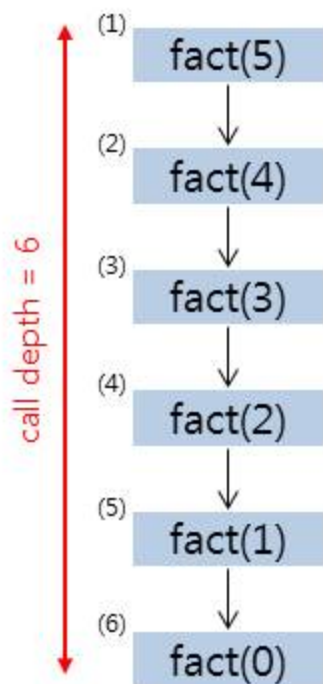
$$\text{fibo}(2) = 1 + 0 = 1$$

$$\text{fibo}(0) = 0$$

$$\text{fibo}(0) = 0$$

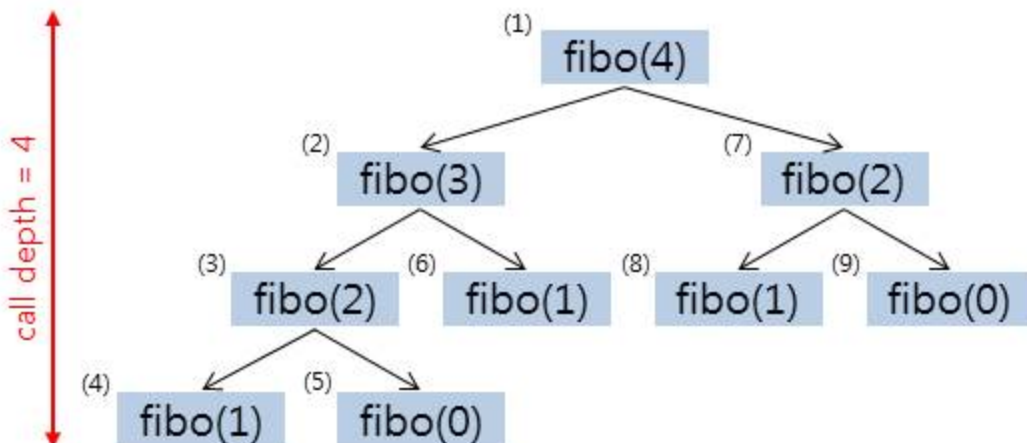
재귀 호출 순서 분석

Call Sequence



call number = 6

Factorial



call number = 9

Fibonacci

참고

- Call Depth: 함수의 호출 깊이. 필요한 메모리(stack)량과 관련 있음.
- Call Number: 함수의 호출 횟수. 계산을 위해 필요한 시간과 관련 있음.

재귀 호출 과정

