Clickseo.com

C Programming

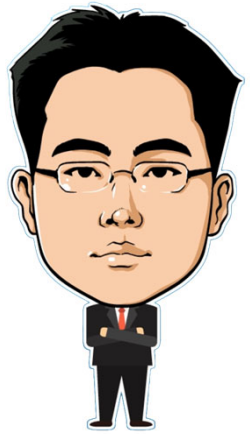
함수

(Functions)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



목 차

- 함수의 이해

백문이불여일타(百聞而不如一打)

- 저장 공간 분류
- 함수와 포인터
- 재귀 함수
- C 표준 라이브러리

함수의 이해

- **함수의 이해**
  - 사용자 정의 함수
- 저장 공간 분류
- 함수와 포인터
- 재귀 함수
- C 표준 라이브러리

백문이불여일타(百聞而不如一打)

함수의 이해 (1/2)

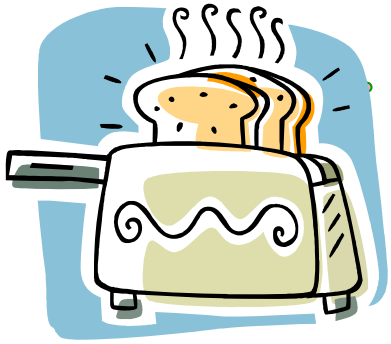
- **함수란 무엇인가?**

"함수는 필요한 데이터를 주면 정해진 행동을 하고
원하는 값을 만들어서 돌려준다"

"잘 정의된 일을 처리하는 기본 단위"

데이터만 주면,
내가 그 모든 값의 평균을
구해주지

나에게 식빵을 주면,
구운 식빵을 만들어 드리죠!!



나에게 정수 2개만 주면,
그 곱을 만들어 주겠어!!

함수의 이해 (2/2)

● 왜 함수를 사용하는가?

❍ 분할과 정복(Divide-and Conquer)

- 어떤 문제를 해결하기 위해 여러 개의 작은 문제로 쪼개는 것

❍ 프로그램의 작성이 용이

- 반복적인 일을 수행하는 경우 원시파일의 크기를 줄일 수 있다.

❍ (장점) 함수 단위로 프로그램 구성 시...

- **모듈화** : 함수 호출을 통한 프로그램 간략화
  - **함수 재 사용**을 통한 프로그램 구성의 편리성
- **표준 함수 이용**을 통한 프로그램 구현의 용이성
- **함수 단위 데이터 접근** 방법(변수의 지역성)을 통해 자료에 대한 제어의 용이성

함수의 이해

사용자 정의 함수

사용자 정의 함수 (1/5)

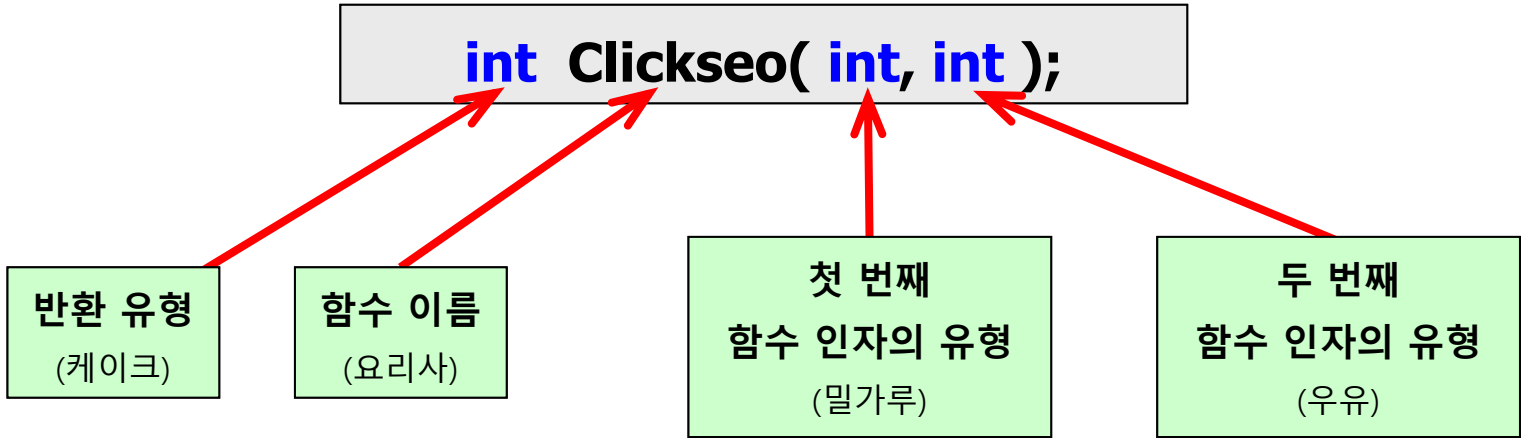
● 사용자 정의 함수 : 함수 원형 선언 및 정의

1. 함수 원형 선언

- 컴파일러에게 함수의 이름과 반환 값의 유형(Return Type) 그리고 함수 인자의 유형들을 알려준다.

2. 함수 정의

- 함수에 대한 코드를 생성하는 작업
- 함수의 구성요소(함수 헤더, 함수 몸체)

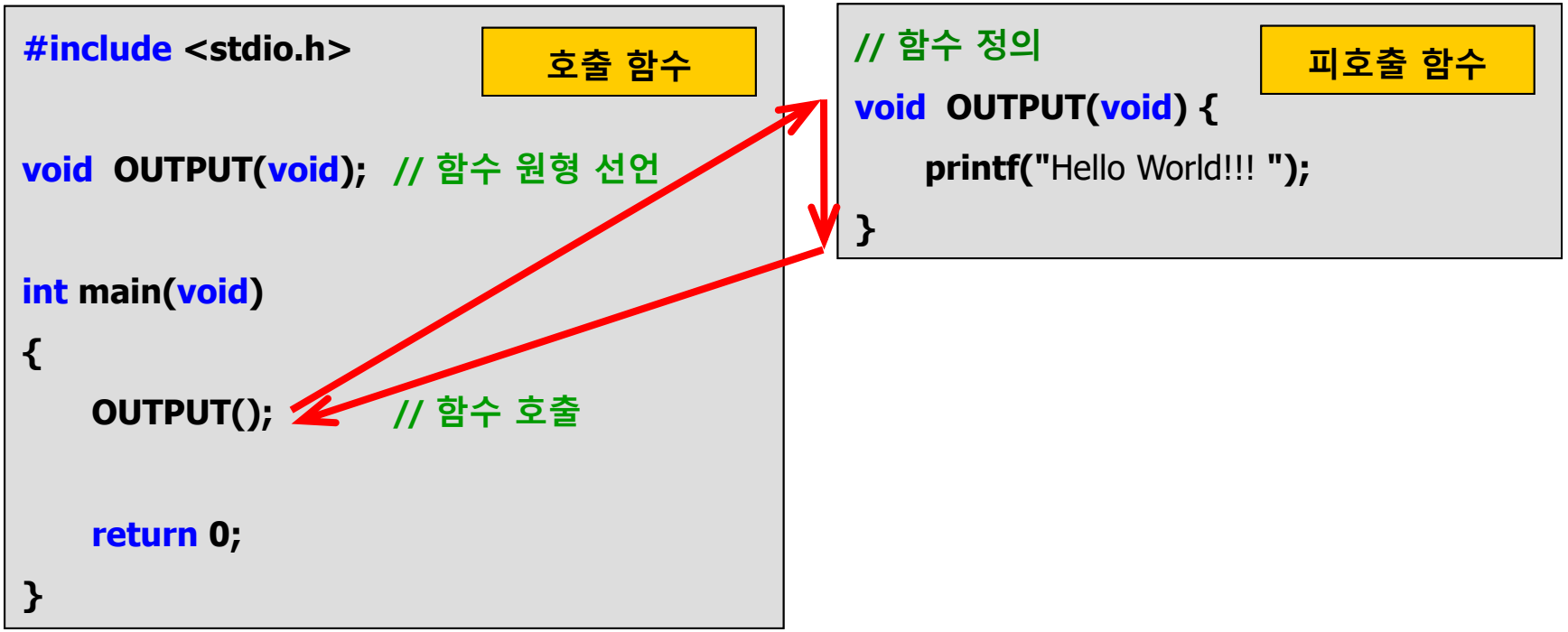


사용자 정의 함수 (2/5)

- **사용자 정의 함수** : 함수 호출

3. 함수 호출

- 호출 함수와 피호출 함수



사용자 정의 함수 (3/5)

- 단일 함수로 프로그램 작성

```
#include <stdio.h>
int main(void)
{
        int         a, b, sum;

        scanf_s("%d %d", &a, &b);

        sum = a + b;

        printf("%d + %d = %d₩n", a, b, sum);

        return 0;
}
```

a, b의 두 정수를 더하는 명령행을 하나의 함수로 작성

사용자 정의 함수 (4/5)

- 여러 개의 함수로 프로그램 작성

```
#include <stdio.h>

int  ADD( int,  int );

int main(void)
{
        int         a, b, sum;

        scanf_s("%d %d", &a, &b);

        sum = ADD( a, b );

        printf("%d + %d = %d₩n", a, b, sum);

        return 0;
}
```

```
int ADD( int a, int b )  {
        int         sum;

        sum = a + b;

        return sum;
}
```

사용자 정의 함수 (5/5)

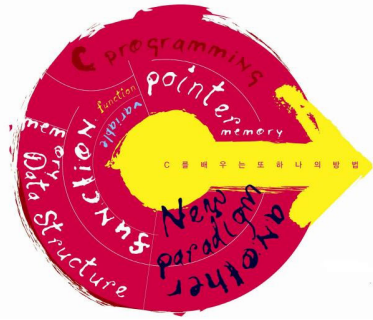
예제 5-1 : 사용자 정의 함수 -- ADD 함수

```
#include <stdio.h>

// 함수 원형 선언
int ADD(int, int);

int main(void)
{
        int     a, b, sum;

        scanf_s("%d %d", &a, &b);              // scanf("%d %d", &a, &b);

        // 함수 호출 : ADD 함수
        sum = ADD( a, b );

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

        return 0;
}

// 함수 정의 : ADD 함수
int ADD(int a, int b) {
        int       sum = a + b;

        return sum;                  // return a + b;
}
```

저장 공간 분류

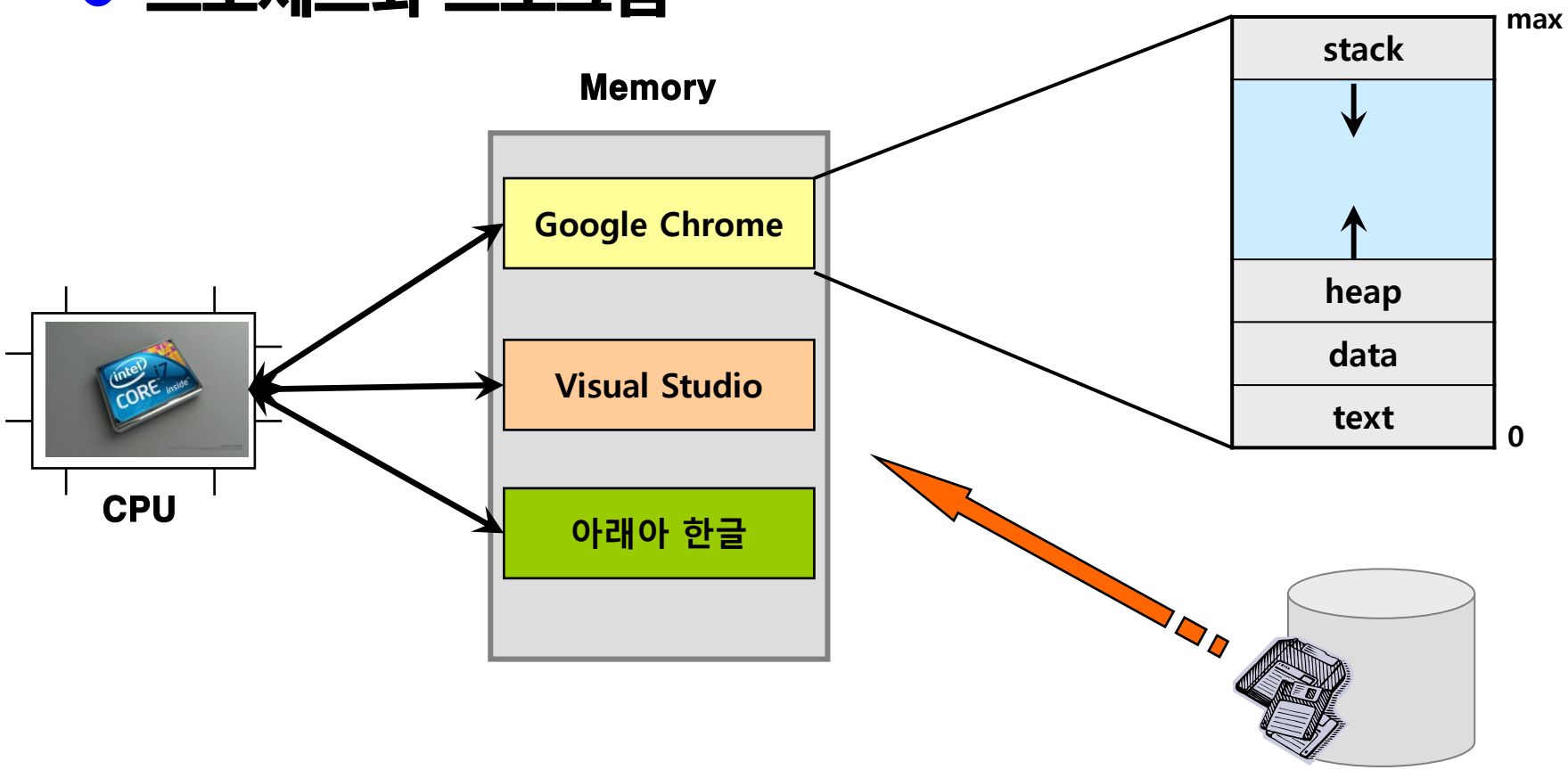
- 함수의 이해
- **저장 공간 분류**
  - 지역 변수와 전역 변수
  - 자동 변수와 정적 변수
  - 외부 변수
  - 레지스터 변수
- 함수와 포인터
- 재귀 함수
- C 표준 라이브러리

백문이불여일타(百聞而不如一打)



저장 공간 분류 (1/2)

- **프로세스와 프로그램**



프로세스 : 운영체제에서 프로세스는 **"실행중인 프로그램"**

프로그램 : 컴퓨터를 실행시키기 위해 차례대로 작성된 **"명령어 집합"**

저장 공간 분류 (2/2)

- 기억 장소 활용에 따른 변수의 종류

<table>
<tr><th>변수의 종류</th><th>예약어</th><th>생존 기간</th><th>유효 범위</th><th>초기화</th><th>초기화 값</th></tr>
<tr><td>자동 변수</td><td>(auto)</td><td rowspan="2">일시적</td><td rowspan="2">지역적</td><td rowspan="2">수행 시</td><td rowspan="2">임의 값</td></tr>
<tr><td>레지스터 변수</td><td>register</td></tr>
<tr><td>(내부) 정적 변수</td><td rowspan="2">static</td><td rowspan="3">영구적</td><td>지역적</td><td rowspan="3">컴파일 시</td><td rowspan="3">0</td></tr>
<tr><td>(외부) 정적 변수</td><td rowspan="2">전역적</td></tr>
<tr><td>외부 변수</td><td>(extern)</td></tr>
</table>

지역 변수와 전역 변수 (1/5)

● 지역 변수(Local Variable)

❍ 함수 또는 블록 안에 정의된 변수

- 사용 범위가 함수 내부로 제한(블록 안에서만 참조 가능)
- **함수 호출 시 생성**(메모리 할당)**되고, 함수 종료 시 소멸**(메모리 반납)
  - 자동변수(automatic variable)
- **서로 다른 함수에 같은 변수 이름 사용 가능**
- 예 : **auto**, **register**, 함수 내부에서 선언 된 **static**

❍ 지역 변수의 초기화

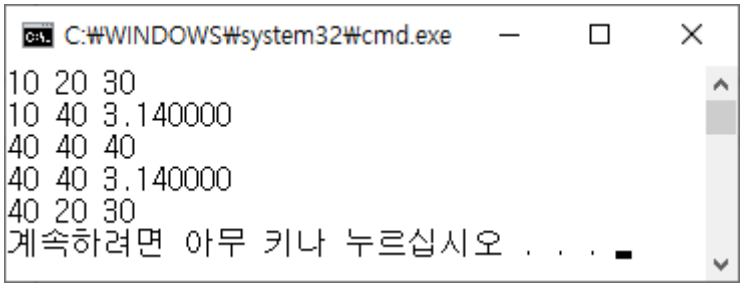
- **함수가 호출될 때마다 메모리를 할당 받고, 종료 시 메모리 반납**
  - 초기화를 하지 않은 지역 변수는 임의의 값을 가진다.

"지역 변수는 함수가 호출될 때 메모리 상에 올라갔다가(메모리 할당),
함수가 종료되면 메모리상에서 사라진다(메모리 반납)."

지역 변수와 전역 변수 (2/5)

예제 5-2 : 지역 변수

```
#include <stdio.h>
int main(void)
{
        int     a = 10, b = 20, c = 30;

        printf("%d %d %d\n", a, b, c );
        {
                int     b = 40;
                double  c = 3.14;

                printf("%d %d %f\n", a, b, c );

                a = b;

                {
                        int     c;

                        c = b;

                        printf("%d %d %d\n", a, b, c );
                }
                printf("%d %d %f\n", a, b, c );
        }
        printf("%d %d %d\n", a, b, c );

        return 0;
}
```



지역 변수와 전역 변수 (3/5)

● 전역 변수(Global Variable)

❍ 모든 함수가 함께 사용(공유)하는 변수

- 함수 외부에서 선언
- 프로그램 시작 시 생성(메모리 할당)되고, 프로그램 종료 시 소멸(메모리 반납)
- 같은 이름의 전역 변수는 하나 이상 사용할 수 없다.
- 예 : **extern**, 함수 외부에서 선언 된 **static**

❍ 전역 변수의 초기화

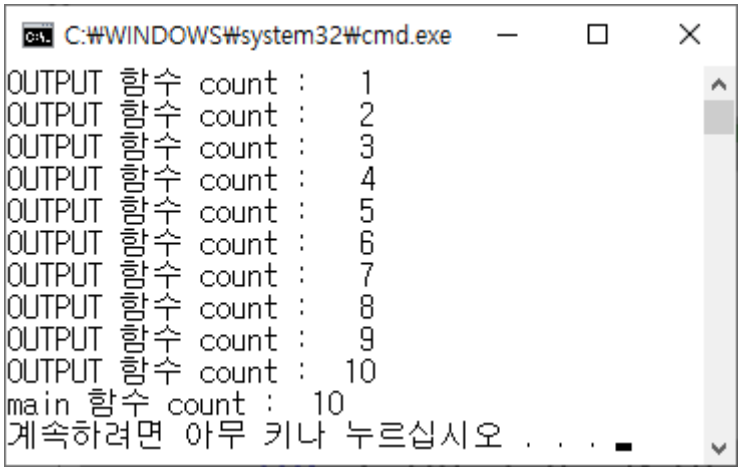
- 프로그램 시작 시 초기화되고, 프로그램 종료 시까지 값을 유지한다.
  - 초기값을 지정하지 않은 경우 0 으로 자동 초기화된다.

"전역 변수와 정적 변수의 사용은 최대한 피해야 한다"

지역 변수와 전역 변수 (4/5)

예제 5-3 : 전역 변수

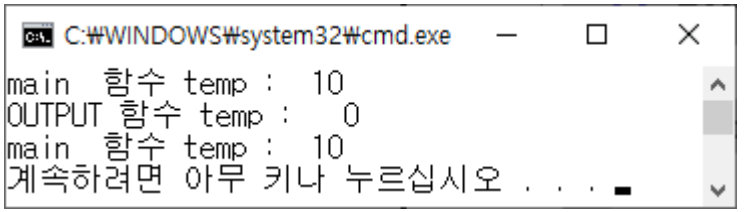
```
#include <stdio.h>

int      count;          // 전역변수

void OUTPUT(void);

int main(void)
{
        for( int i=0; i<10; i++ )
                OUTPUT();
        printf("main 함수 count : %3d \n", count );

        return 0;
}

void OUTPUT(void) {
        count++;
        printf("OUTPUT 함수 count : %3d \n", count );
}
```



지역 변수와 전역 변수 (5/5)

예제 5-4 : 동일한 이름의 전역 변수와 지역 변수

```
#include <stdio.h>

int     temp;   // 전역변수 : temp

void OUTPUT(void);

int main(void)
{
        int     temp = 10;      // 지역변수 : temp

        printf("main  함수 temp : %3d \n", temp );
        OUTPUT();
        printf("main  함수 temp : %3d \n", temp );

        return 0;
}

void OUTPUT(void) {
        printf("OUTPUT 함수 temp : %3d \n", temp );
}
```



"지역 내에서는 지역 변수가 전역 변수보다 우선 시 된다."

저장 공간 분류

자동 변수, 정적 변수

자동 변수와 정적 변수 (1/3)

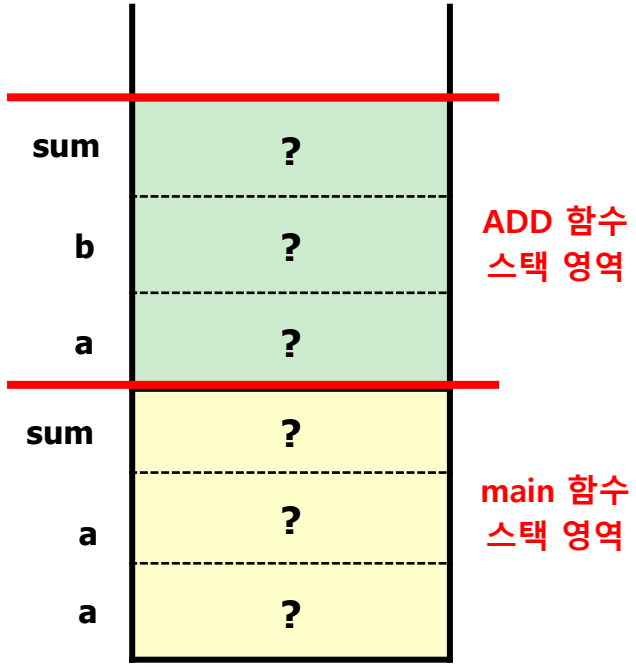
● 자동 변수(Auto Variable)

```
auto  자료형  변수명;
```

❍ 선언된 블록 범위 안에서 자동 생성되고 자동 반납된다.

- 지역 변수의 또 다른 표현
  - 예 : 함수 내부에서 선언되는 지역변수, 매개변수
- **스택 사용** : 선언만 하고 초기화 하지 않으면 임의의 값을 갖는다.

```
void main(void)
{
        auto  int     a,  b, sum;
        sum  = ADD(a, b);
        return ;
}

void ADD(int a, int b) {
        int           sum;
        sum  = a + b;
        return sum;
}
```



자동 변수와 정적 변수 (2/3)

● 정적 변수(Static Variable)

```
static 자료형 변수명;
```

❍ 프로그램 종료 전까지 할당 받은 메모리 공간을 유지한다.

- 정적 변수의 초기화
  - 프로그램 시작 시 초기화되며, 프로그램 종료 시 까지 유지된다.
  - 초기값을 지정하지 않을 경우 자동 초기화 된다.

❍ 지역 정적 변수

- **함수 내부에서 선언되며, 해당 함수 안에서 사용 되는 지역변수**
  - **지역 변수의 값을 프로그램 종료 시 까지 유지하기 위해 사용**
  - 함수 종료 후에도 소멸(메모리 반납)되지 않으며, 다시 함수 호출 시 그 직전의 값을 참조

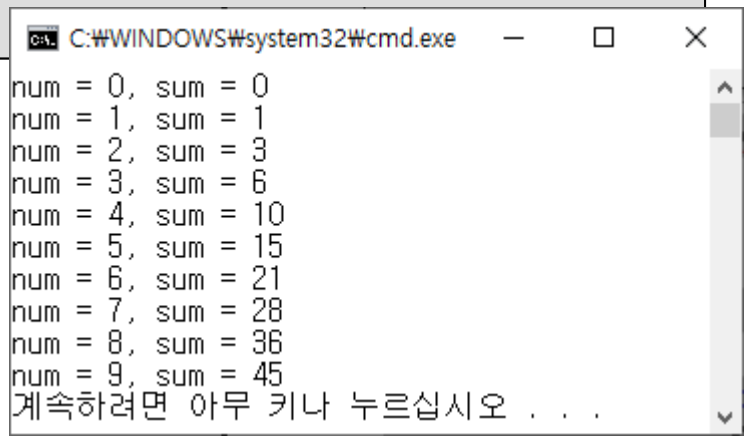
❍ 전역 정적 변수

- 함수 외부에서 선언되며, 전역변수로 사용 되는 정적 변수
  - **정적 변수를 전역 변수로 사용하는 경우 다른 파일에서 접근 불가**

자동 변수와 정적 변수 (3/3)

예제 5-5 : 정적 변수

```
#include <stdio.h>

void SUM( int );

int main(void)
{
        for( int i=0; i<10; i++ )
                SUM( i );

        // Error : 'sum' : undeclared identifier
        // printf("i = %d, sum = %d \n", i, sum);

        return 0;
}

void SUM( int num ) {
        static int      sum = 0;        // (지역)정적변수 : 처음 함수 호출 시 한 번만 초기화

        sum += num;

        printf("num = %d, sum = %d \n", num, sum);
}
```



저장 공간 분류

외부 변수, 레지스터 변수

외부 변수

● 외부 변수(Extern Variable)

extern 자료형 변수명;

❍ 다른 파일에 정의 된 전역변수를 선언만 하고 사용

- 별도의 기억 장소 할당 없이 기존의 변수와 기억 장소 공유
  - 변수의 영향력이 가장 넓다 : **남용은 금물!!!**

- **외부 변수의 장점**
  - 매개 변수처럼 전달이 불필요하며, 시간이 소비되지 않으므로 보다 효율적이다.
  - 변수들의 종류 중에서 통용 범위가 가장 넓고 생존기간이 영구적이다(모든 파일).

- **외부 변수의 단점**
  - 프로그램을 크고 복잡하게 만들어 부작용의 위험성
  - 함수 인수의 주고 받음이 불분명 : **함수 간의 독립성 상실**
  - 외부 변수를 남용하면 모듈 간의 자료결합이 강하게 되어 프로그램의 구조를 해치게 되므로 생산성이 저하된다 : **함수 간의 강결합성**

1) **프로그램 전체를 총괄하는 변수**
2) **프로그램 전체의 상황을 기억하는 변수**

와 같은 경우에 사용한다(반드시 그러해야 되는 것은 아니지만...).

레지스터 변수 (1/2)

● 레지스터 변수(Register Variable)

❍ 메모리가 아닌 CPU의 레지스터에 저장되는 변수

- 처리 속도가 빠르지만, 사용 가능한 레지스터 개수는 제한된다.
  - 보통 2,3 개 정도만 레지스터 변수로 사용 가능하다.
  - **주의 : 레지스터 변수에 주소 연산자(&)는 사용할 수 없다.**

- **자동변수**와 기능적으로 동일하다.

```
register 자료형 변수명;
```

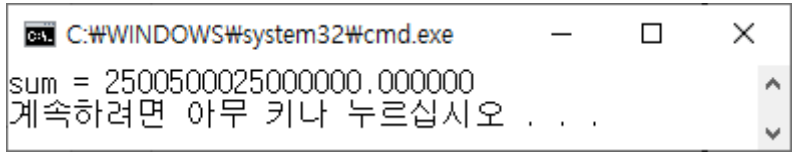
❍ 사용 가능한 데이터 형 또한 char, int, 포인터 형으로 제한된다.

- **외부 변수나 정적 변수는 레지스터 변수로 정의할 수 없다.**

레지스터 변수 (2/2)

예제 5-6 : 레지스터 변수

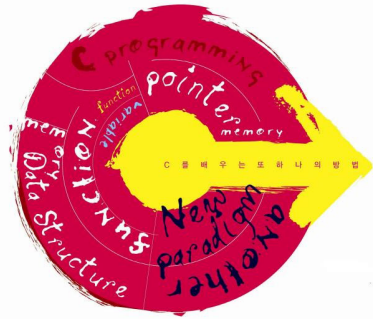
```
#include <stdio.h>
int main(void)
{
        double sum = 0;

        register int      i, j;              // 레지스터 변수
        for( i=0; i<=10000; i++ ) {
                for( j=0; j<=10000; j++ )
                        sum += i * j;
        }

        printf("sum = %f \n", sum);

        return 0;
}
```



함수와 포인터

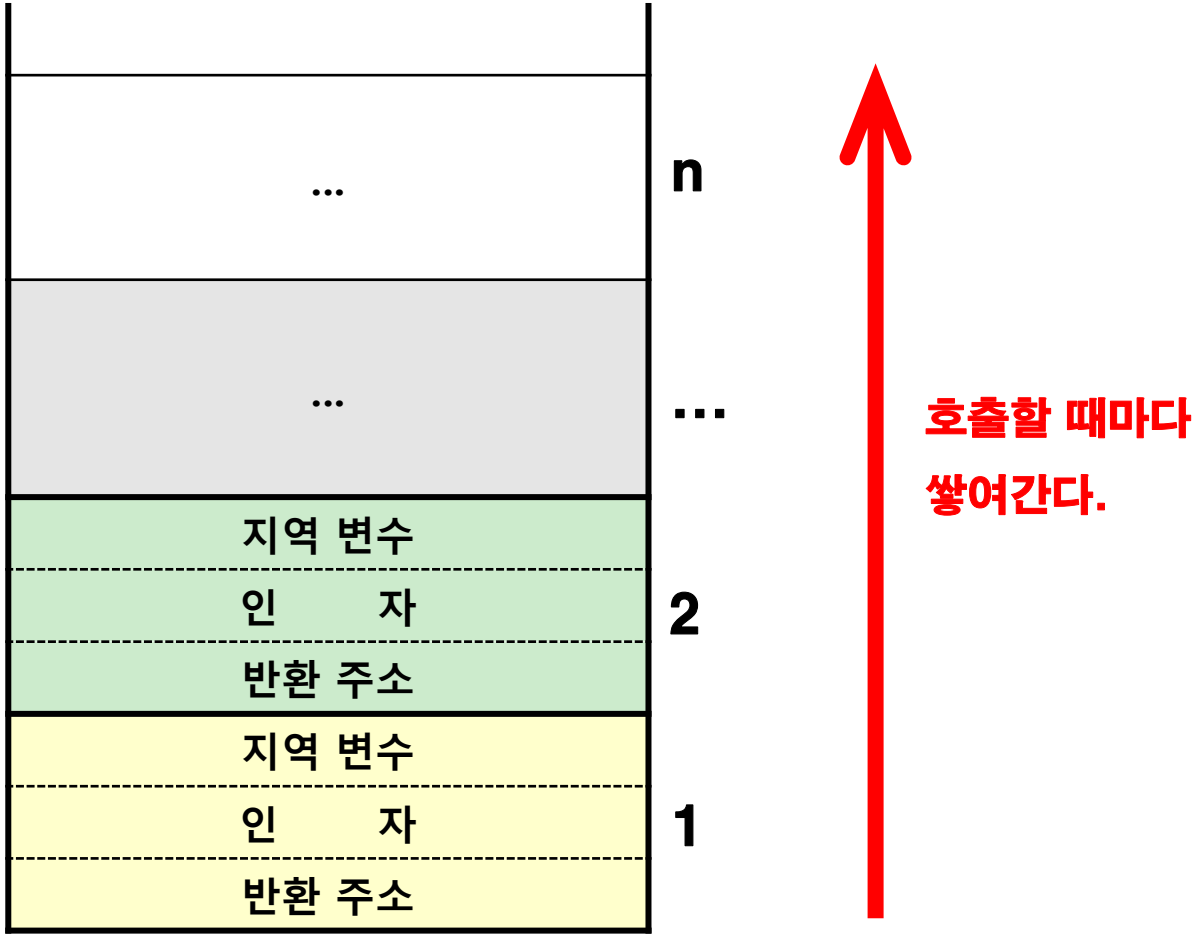
- 함수의 이해
- 저장 공간 분류
- **함수와 포인터**
  - 함수 호출
  - 함수 포인터
- 재귀 함수
- C 표준 라이브러리

백문이불여일타(百聞而不如一打)



함수 호출 (1/4)

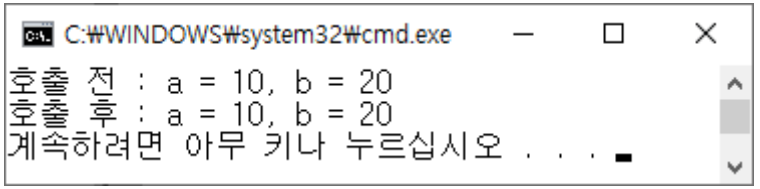
- 함수 호출 시 스택 상태

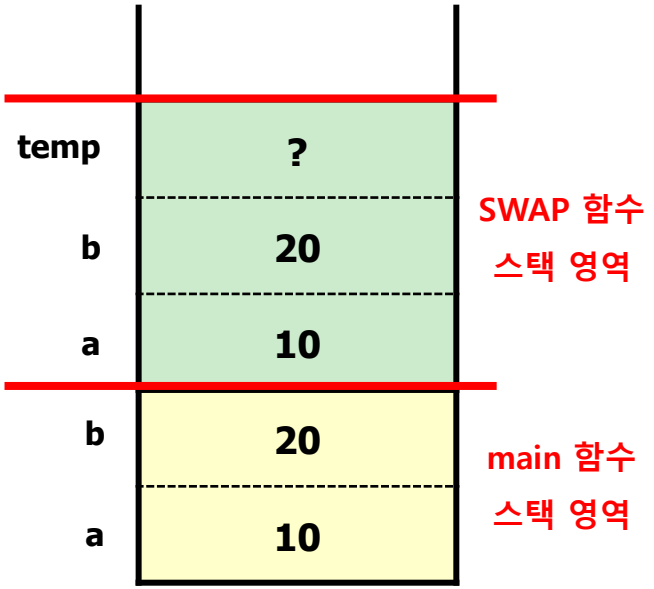


함수 호출 (2/4)

예제 5-7 : 값에 의한 전달 -- pass by Value

```
#include <stdio.h>

void SWAP( int, int );

int main(void)
{
        int     a = 10, b = 20;

        printf("호출 전 : a = %d, b = %d \n", a, b);
        SWAP( a, b );
        printf("호출 후 : a = %d, b = %d \n", a, b);

        return 0;
}

void SWAP ( int a, int b ) {
        int     temp;

        temp = a;
        a = b;
        b = temp;
}
```

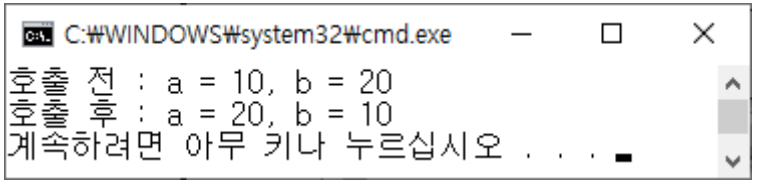


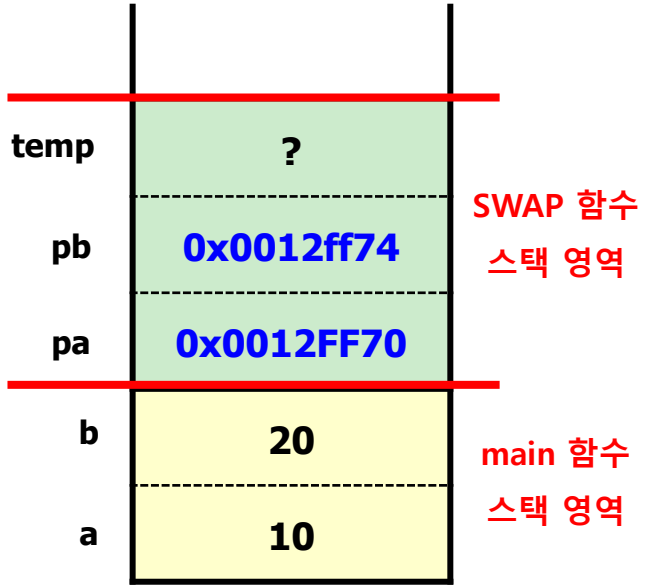


함수 호출 (3/4)

예제 5-8 : 주소에 의한 전달 -- pass by Address

```
#include <stdio.h>

void SWAP( int *, int * );

int main(void)
{
        int     a = 10, b = 20;

        printf("호출 전 : a = %d, b = %d \n", a, b);
        SWAP( &a, &b );
        printf("호출 후 : a = %d, b = %d \n", a, b);

        return 0;
}

void SWAP ( int *pa, int *pb ) {
        int     temp;

        temp = *pa;
        *pa = *pb;
        *pb = temp;
}
```

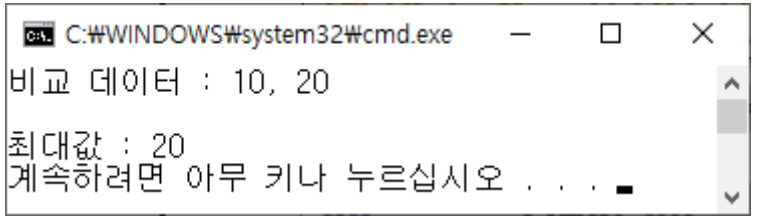


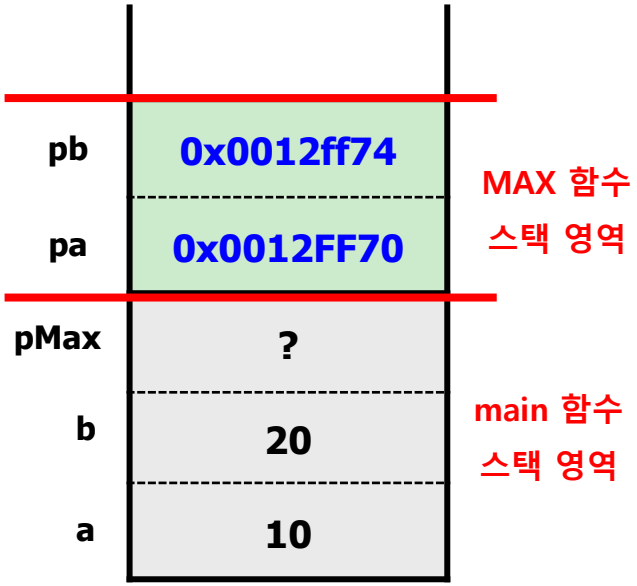


함수 호출 (4/4)

예제 5-9 : 포인터(메모리 주소)를 반환 값으로 갖는 함수

```
#include <stdio.h>

int     *MAX( int *, int * );

int main(void)
{
        int     a = 10, b = 20;
        int     *pMax;

        pMax = MAX( &a, &b );

        printf("비교 데이터 : %d, %d \n", a, b);
        printf("\n최대값 : %d \n", *pMax );

        return 0;
}

int     *MAX( int *pa, int *pb ) {
        return  *pa > *pb ? pa : pb;
}
```





함수와 포인터

함수 포인터

함수 포인터 (1/3)

- **함수 포인터(Function Pointer) : 간접 호출**
  - **함수를 가리킬 수 있는 포인터**
    - 자주 사용되는 함수의 주소를 배열에 저장해 두고 호출하면 속도가 빨라진다.
    - 잘 쓰이지는 않지만, 수치해석이나 그래픽 같은 분야에서 사용

```
#include <stdio.h>

int ADD( int, int );

int main(void)
{
        int a = 10, b = 20, sum = 0;
        int (*p)(int, int) = ADD;     // 함수 포인터 변수 선언 및 초기화

        sum = p(a, b);     // 함수 포인터를 이용한 함수 호출(간접 호출)
        printf("%d + %d = %d \n", a, b, sum);

        return 0;
}

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

함수 포인터 (2/3)

예제 5-10 : 함수 포인터와 사칙 연산 계산 함수 (1/2)

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

int     ADD( int, int );
int     SUB( int, int );
int     MUL( int, int );
int     DIV( int, int );
int     CAL( int, int, int(*)(int, int) );

int main(void)
{
        char    op;
        int     a, b, res;

        printf("수식 입력(예: 10 + 20) : ");
        scanf_s("%d %c %d", &a, &op, sizeof(op), &b);
        // scanf("%d %c %d", &a, &op, &b);

        switch( op )   {
                case '+': res = CAL( a, b, ADD );       break;
                case '-': res = CAL( a, b, SUB );       break;
                case '*': res = CAL( a, b, MUL );       break;
                case '/': res = CAL( a, b, DIV );       break;
                default: printf("지원되지 않는 연산자!!! \n");
                        return 0;
        }

        printf("수식 결과 : %d %c %d = %d \n", a, op, b, res);
        return 0;
}
```

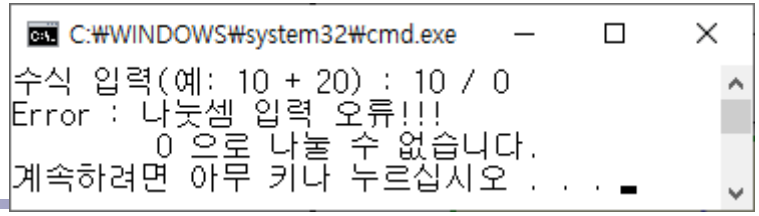
```
C:₩WINDOWS₩system32₩cmd.exe
수식 입력(예: 10 + 20) : 10 + 20
수식 결과 : 10 + 20 = 30
계속하려면 아무 키나 누르십시오 . . .
```

함수 포인터 (3/3)

예제 5-10 : 함수 포인터와 사칙 연산 계산 함수 (2/2)

```
int     CAL( int a, int b, int(*funcPtr)(int, int) ) {
        int     res;

        // 함수 포인터를 이용한 간접 호출
        res = funcPtr( a, b );

        return res;
}
```

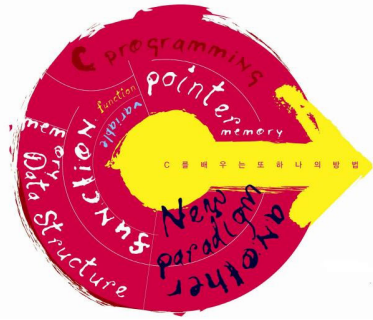
```
int ADD( int a, int b )      {      return a + b;      }
int SUB( int a, int b )      {      return a - b;      }
int MUL( int a, int b )      {      return a * b;      }

int DIV( int a, int b ) {
        if( b == 0 )   {
                printf("Error : 나눗셈 입력 오류!!! \n");
                printf("\t 0 으로 나눌 수 없습니다. \n");
                exit(0);
        }

        return a / b;
}
```



재귀 함수

- 함수의 이해
- 저장 공간 분류
- 함수와 포인터
- **재귀 함수**
  - 반복적 용법과 재귀적 용법
  - 동적 프로그래밍
- C 표준 라이브러리

백문이불여일타(百聞而不如一打)



재귀 함수 (1/3)

● 재귀 함수(Recursive Function)

❍ 자기 자신의 함수를 호출 함으로써, 반복적인 처리를 하는 함수

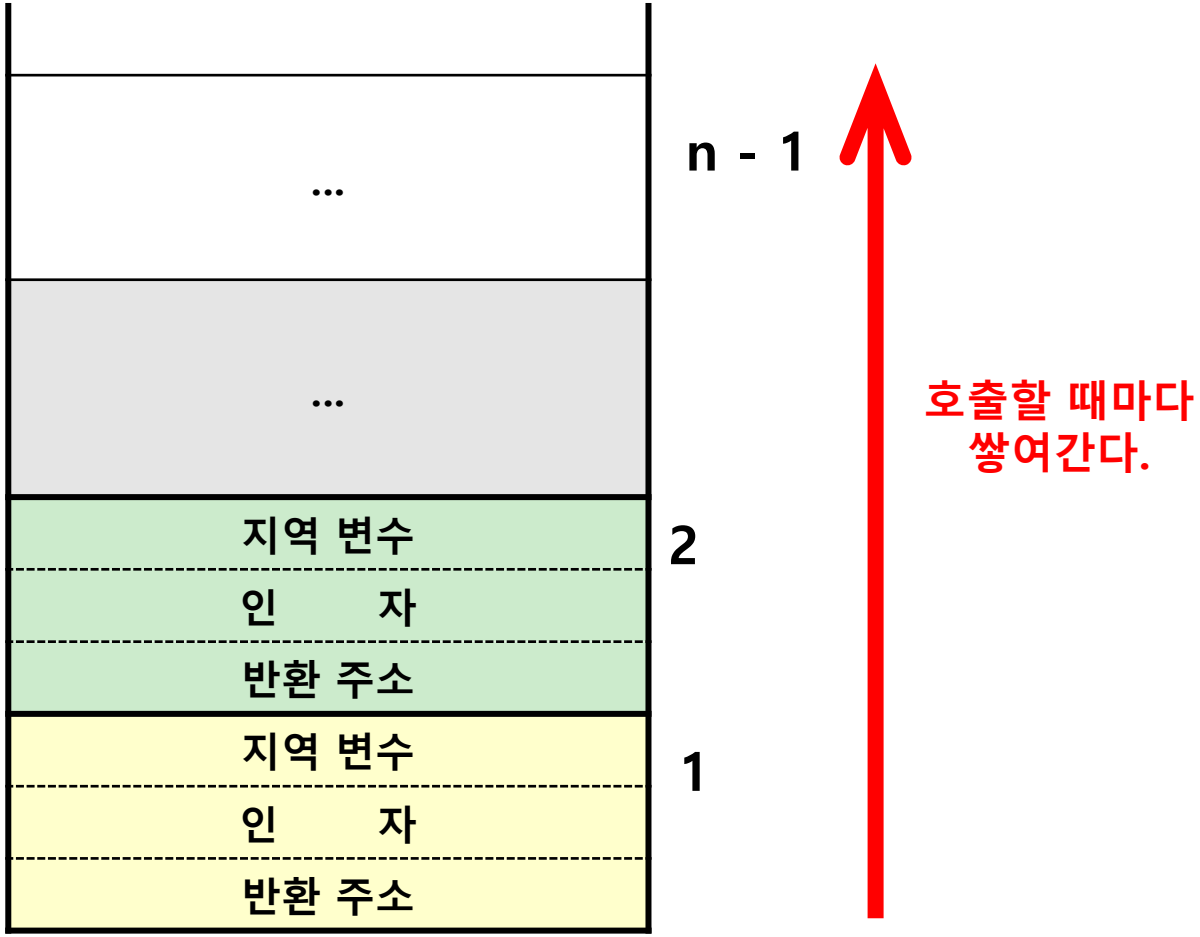
- 재귀 함수 안에서 사용하는 변수는 **지역변수**(자동변수)
- 재귀 함수의 인수들은 **값에 의한 전달**(pass by Value) 방식으로 전달된다.
- **주의 : 반드시 탈출**(종료) **조건 명시!!!**
  - **Stack Overflow 오류 발생 주의!!!**

- **장점**
  - 코드가 훨씬 간결 해지며, 프로그램을 보기가 쉽다.
  - 또한 프로그램 오류 수정이 용이하다.

- **단점**
  - 코드 자체를 이해하기 어렵다.
  - 또한 메모리 공간을 많이 요구한다.

재귀 함수 (2/3)

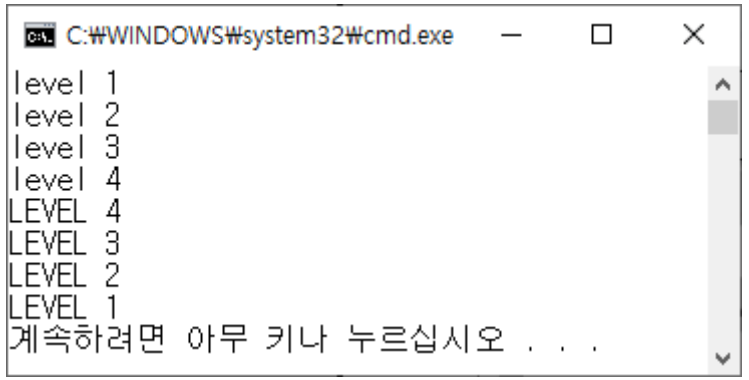
- 재귀 함수 호출 시 스택 상태



재귀 함수 (3/3)

예제 5-11 : 재귀 함수

```
#include <stdio.h>

void OUTPUT( int );

int main(void)
{
        OUTPUT( 1 );

        return 0;
}

void OUTPUT(int num) {                  // 재귀 함수
        printf("level %d\n", num );
        if(num < 4)                     // 재귀 함수 탈출(종료) 조건
                OUTPUT( num+1 );
        printf("LEVEL %d\n", num );

        return;
}
```



재귀 함수

반복적 용법과 재귀적 용법

반복적 용법과 재귀적 용법 (1/3)

- **반복적 정의**
  - 반복 함수가 반복적으로 정의된다.
    - 함수 정의는 매개변수를 포함하나 함수 자체는 포함하지 않는다.

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1) * (n-2) \ldots 3 * 2 * 1 & \text{if } n > 0 \end{cases}$$

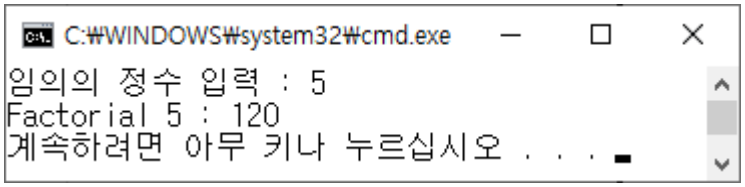
- **재귀적 정의**
  - 함수가 자기 자신을 포함한다.

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * \text{factorial}\ (n-1) & \text{if } n > 0 \end{cases}$$

반복적 용법과 재귀적 용법 (2/3)

예제 5-12 : 재귀 함수 -- 반복적 용법

```
#include <stdio.h>

int Factorial( int num );

int main(void)
{
        int     num;

        printf("임의의 정수 입력 : ");
        scanf_s("%d", &num);          // scanf("%d", &num);

        printf("Factorial %d : %d \n", num, Factorial( num ) );

        return 0;
}
```

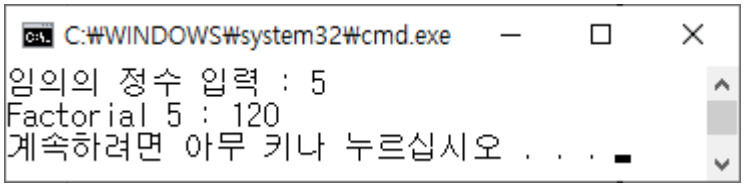



```
int Factorial( int num ) {
        int     res = 1;

        for ( int i = 1; i <= num; i++ )
                res = res * i;

        return res;
}
```

반복적 용법과 재귀적 용법 (3/3)

예제 5-13 : 재귀 함수 -- 재귀적 용법

```
#include <stdio.h>

int Factorial( int num );

int main(void)
{
        int     num;

        printf("임의의 정수 입력 : ");
        scanf_s("%d", &num);          // scanf("%d", &num);

        printf("Factorial %d : %d \n", num, Factorial(num) );

        return 0;
}
```




```
int Factorial( int num ) {
        if(num == 0)
                return 1;

        return num * Factorial (num - 1);
}
```

재귀 함수

동적 프로그래밍

동적 프로그래밍 (1/2)

● 재귀적 해법

❍ 큰 문제에 닮음 꼴의 작은 문제가 깃든다

❍ 잘 쓰면 보약, 잘못 쓰면 맹독

- 관계중심으로 파악함으로써 문제를 간명하게 볼 수 있다.
- **재귀적 해법을 사용하면 심한 중복 호출이 일어나는 경우가 있다.**

❍ 재귀적 해법이 바람직한 예

- **계승**(factorial) 구하기
- 퀵 정렬, 병합 정렬 등의 정렬 알고리즘
- 그래프의 깊이 우선 탐색(DFS, Depth First Search)

❍ 재귀적 해법이 치명적인 예

- **피보나치 수** 구하기
- 행렬 곱셈 최적순서 구하기

동적 프로그래밍 (2/2)

● 동적 프로그래밍의 적용 조건

❍ 최적 부분구조(optimal substructure)

- 큰 문제의 최적 솔루션에 작은 문제의 최적 솔루션이 포함된다.

❍ 재귀 호출 시 중복(overlapping recursive calls)

- 재귀적으로 구현했을 때 중복 호출로 심각한 비효율이 발생한다.

동적 프로그래밍이 그 해결책 !!!

동적 프로그래밍 : 피보나치 수열 (1/4)

- **피보나치 수열**
  - **피보나치**(Fibonacci)
    - 1,200년 경에 활동한 이탈리아 수학자

"아주 간단한 문제지만...
동적 프로그래밍의 동기와 구현이 다 포함되어 있다."

"토끼 한 마리가 매년 새끼 한 마리를 낳는다.
새끼는 한 달 후부터 새끼를 낳기 시작한다.
최초 토끼 한 마리가 있다고 하면...
한 달 후에 토끼는 두 마리가 되고 두 달 후에는 세 마리가 되고..."

$$f_n = f_{n-1} + f_{n-2} (n \geq 3)$$
$$f_1 = f_2 = 1 (n = 1,2)$$

동적 프로그래밍 : 피보나치 수열 (2/4)

- **피보나치 수열** : 재귀적 용법

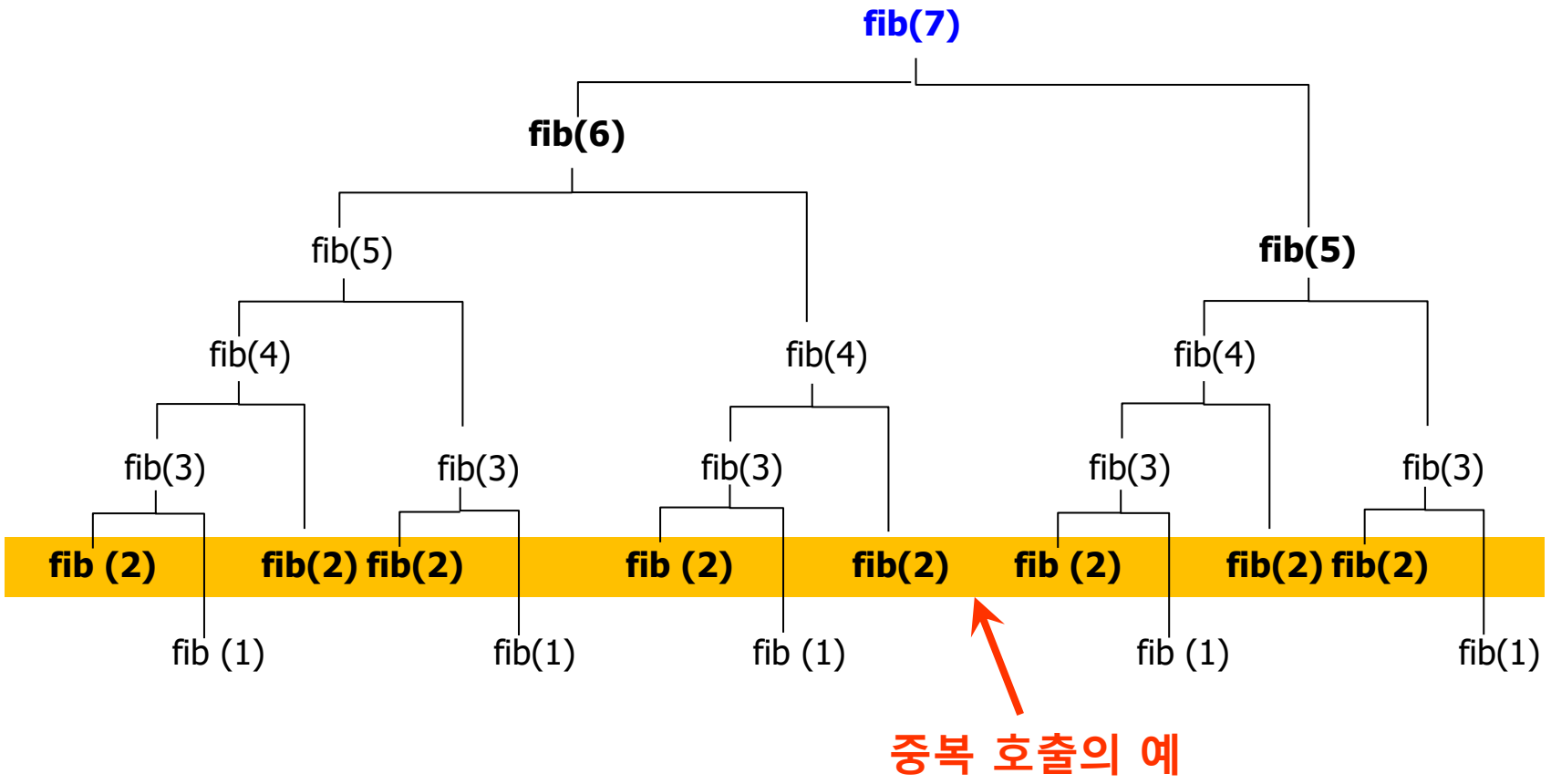
```
Fibonacci(num)
{
        if ( num = 1 or num = 2 )
                then  return  1;
        else
                return ( Fibonacci(num - 1) + Fibonacci(num - 2) );
}
```

"엄청난 중복 호출이 존재한다."

--> 재귀적 알고리즘은 지수함수에 비례하는 시간이 든다.

동적 프로그래밍 : 피보나치 수열 (3/4)

- **피보나치 수열** : 재귀적 용법
  - 문제점 : 중복 호출!!!



동적 프로그래밍 : 피보나치 수열 (4/4)

- **피보나치 수 구하기** : 동적 프로그래밍 알고리즘

```
Fibonacci(num)
{
    f[1] ← f[2] ← 1;

    for i ← 3 to num
        f[i] ← f[i - 1] + f[i - 2];

    return f[num];
}
```

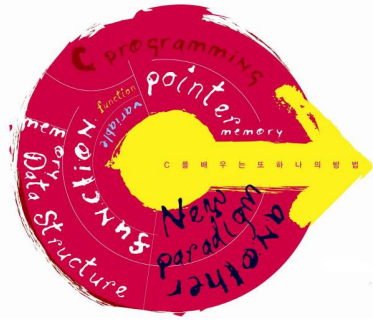
fibo

1	1	?	?	?	?	?

C 표준 라이브러리

- 함수의 이해
- 저장 공간 분류
- 함수와 포인터
- 재귀 함수
- **C 표준 라이브러리**
  - 표준 라이브러리 : <stdlib.h>
  - 날짜 및 시간 관련 함수 : <time.h>
  - 수학 관련 함수 : <math.h>

백문이불여일타(百聞而不如一打)



C 표준 라이브러리 (1/2)

- **표준 라이브러리 함수(Standard Library Functions)**
  - C/C++ 언어에서 프로그래머의 편의성을 도모하기 위해 프로그램 언어 개발자들에 의해 작성되어 포함된 함수
    - **C 표준 라이브러리**(C standard library)
      - **ISO C 라이브러리**
      - **C POSIX 라이브러리** 와 거의 동시에 개발
    - **glibc**(The GNU C Library)
  - **#include**
    - 표준 라이브러리 함수를 불러 들이기 위해 전 처리 구문 영역에 사용
    - 예 : **#include <stdio.h>**
      - 대표적인 표준 라이브러리 함수인 **scanf** 와 **printf** 계열 함수를 사용하기 위한 헤더 파일

C 표준 라이브러리 (2/2)

C 표준 라이브러리(C Standard Library)

헤더 파일

- 하나 이상의 함수 원형 선언과 자료형의 정의 그리고 다양한 매크로들을 포함
- **현재 총 29개의 헤더 파일을 제공**

- **1995년, 규범 별첨 1(NA1)에서 3개의 헤더 파일 추가**
  - iso646.h, wchar.h 그리고 wctype.h

- **1999년, C99 버전에서 새롭게 6개의 헤더 파일 추가**
  - complex.h, fenv.h, inttypes.h, stdbool.h, stdint.h 그리고 tgmath.h

- **2011년, C11 버전에서 5개의 헤더 파일 추가**
  - stdalign.h, stdatomic.h, stdnoreturn.h, threads.h 그리고 uchar.h

C 표준 라이브러리

표준 라이브러리 : <stdlib.h>

표준 라이브러리 (1/6)

- **데이터 유형(Data Types)**

```
#include <stdlib.h>

typedef unsigned int size_t;
typedef unsigned short wchar_t;
```

- **매크로(Macro) : 변수 및 상수 그리고 매크로 함수**

```
#include <stdlib.h>

#define NULL        ((void *) 0)

// exit 함수를 위해 정의된 값
#define EXIT_SUCCESS   0
#define EXIT_FAILURE   1

// rand 함수 사용 시 임의의 난수 발생 최대값
#define RAND_MAX 0x7fff

#define __max(a, b) (((a) > (b)) ? (a) : (b))
#define __min(a, b) (((a) < (b)) ? (a) : (b))
```

표준 라이브러리 (2/6)

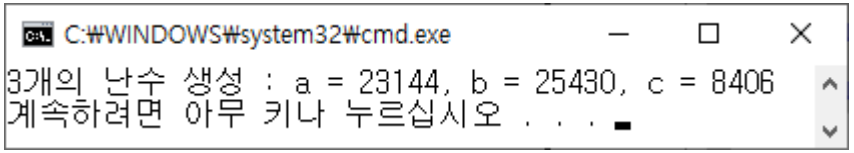
- **임의의 난수 생성 함수**

```
#include <stdlib.h>

// 0 ~ RAND_MAX 사이에서 임의의 난수를 생성하여 반환
int  rand (void);

// seed 있는 랜덤발생 함수, 초기의 seed 는 1이다.
void  srand ( unsigned  int  seed );
```

표준 라이브러리 (3/6)

예제 5-14 : 임의의 난수 생성

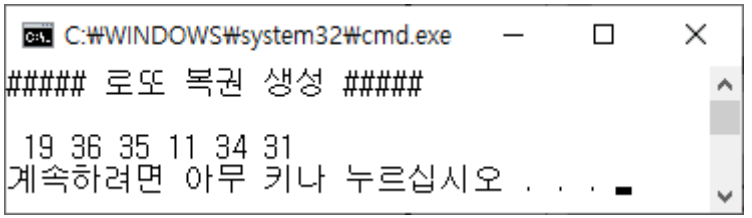
```
#include <stdio.h>
#include <time.h>       // time
#include <stdlib.h>     // srand, rand

int main(void)
{
        int a, b, c;

        srand( (unsigned int) time(NULL) );

        a = rand();
        b = rand();
        c = rand();

        printf("3개의 난수 생성 : a = %d, b = %d, c = %d \n",  a, b, c );

        return 0;
}
```



표준 라이브러리 (4/6)

예제 5-15 : 임의의 난수 생성과 범위 설정

```
#include <stdio.h>
#include <time.h>       // time
#include <stdlib.h>     // rand, srand

int main(void)
{
        int     temp;

        printf("##### 로또 복권 생성 ##### \n\n");
        srand( (unsigned int)time(NULL) );
        for ( int i=1; i<=6; i++ )   {
                temp = rand() % 45 + 1;          // 최소값과 최대값
                printf("%3d", temp );
        }
        printf("\n");

        return 0;
}
```



표준 라이브러리 (5/6)

- **프로세스 제어 관련 함수**

```
#include <stdlib.h>

void  exit( int  exit_code );        //프로그램을 정상적인 상태로 종료
void  abort ();                      // 프로그램을 그 상태에서 정지(비정상적인 종료)

// 프로세스가 exit 함수를 호출하여 종료할 때 수행되는 함수들을 등록한다.
int  atexit (void  (*func)(void));

// exit 함수와 같지만 clean-up-action을 수행하지 않고 프로그램 종료
void  _exit( int  exit_code );
void  _Exit( int  exit_code );                              // C99
```

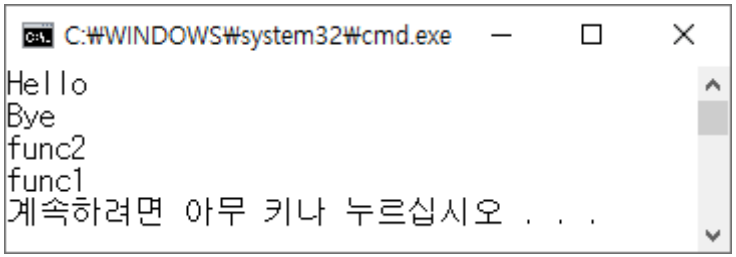
```
#include <stdlib.h>

int  system( const  char  *command );
```

호출 성공 : 0 값을 반환
범위 오류 : -1 값을 반환

표준 라이브러리 (6/6)

예제 5-16 : 프로그램 흐름 제어 관련 함수

```
#include <stdio.h>
#include <stdlib.h>        // exit, atexit

void  func1(void);
void  func2(void);

int main(void)
{
        printf("Hello \n");
        atexit( func1 );
        atexit( func2 );
        printf("Bye \n");

        exit(0);           // _exit(0); 로 수정 시 func1과 func2는 실행되지 않고 종료한다.
}

void func1(void) {
        printf("func1 \n");
}

void func2(void) {
        printf("func2 \n");
}
```



C 표준 라이브러리

날짜 및 시간 관련 함수 : <time.h>

날짜 및 시간 관련 함수 (1/3)

- **데이터 유형(Data Types)**

```
#include <time.h>

typedef  long  time_t;          // 시간을 표현하는 산술 유형
typedef  long  clock_t;         // 프로세스 실행 시간 유형
```

- **매크로(Macro) : 변수 및 상수**

```
#include <time.h>

#define  CLOCKS_PER_SEC  1000
#define  CLK_TCK  CLOCKS_PER_SEC
```

날짜 및 시간 관련 함수 (2/3)

- **시간 조작 함수**

```
#include <time.h>

// 프로그램 시작 후의 경과한 clock tick(1초에 18.2만큼 증가)를 반환
clock_t clock();
```

호출 성공 : process tick count 값을 반환
호출 실패 : (clock_t) -1 값을 반환

```
#include <time.h>

// 1970년 1월 1일 자정부터 경과된 현재 시간을 초 단위로 계산하여 반환
time_t time( time_t *time );
```

호출 성공 : 현재 시간 값을 반환
호출 실패 : (time_t) -1 값을 반환

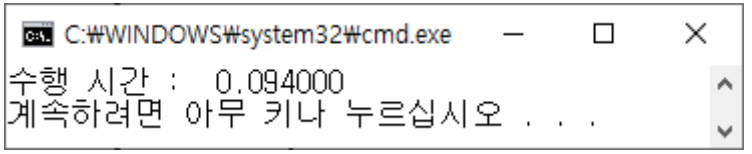
```
#include <time.h>

// time2 – time1 을 초로 계산하여 결과 값을 반환
double difftime( time_t time2, time_t time1 );
```

호출 성공 : 두 시간의 차를 계산한 실수 값을 반환

날짜 및 시간 관련 함수 (3/3)

예제 5-17 : 프로그램 수행 시간

```
#include <stdio.h>
#include <time.h>       // clock_t, CLK_TCK, clock 함수

int main(void)
{
        clock_t         start, end;
        double          time;

        start = clock();
        for(int i=0; i<30000000; i++)
                ;
        end = clock();

        time = (double)(end - start) / CLK_TCK;
        printf("수행 시간 : %lf \n", time );

        return 0;
}
```



C 표준 라이브러리

수학 관련 함수 : <math.h>

수학 관련 함수 (1/13)

- **수학 계산 함수 : abs**

```
#include <math.h>

// 정수형의 절대값 을 계산하여 반환
int  abs( int  num );
long  labs( long  num );
long  long  llabs( long  long  num );                    // C99
```

```
#include <math.h>

// 부동 소수점의 절대값을 계산하여 반환
float  fabsf( float  arg );                              // C99
double  fabs( double  arg );
long  double  fabsl( long  double  arg );                // C99
```

수학 관련 함수 (2/13)

- **수학 계산 함수** : fmod, fmax, fmin

```
#include <math.h>

// 실수 x 를 실수 y 로 나눈 나머지를 계산하여 반환
float  fmodf( float  x,  float  y );                                  // C99
double  fmod( double  x,  double  y );
long  double  fmodl( long  double  x,  long  double  y );             // C99

// 실수 x 와 실수 y 에서 큰 값을 반환
float  fmaxf( float  x,  float  y );                                  // C99
double  fmax( double  x,  double  y );                                // C99
long  double  fmaxl( long  double  x,  long  double  y );             // C99

// 실수 x 와 실수 y 에서 작은 값을 반환
float  fminf( float  x,  float  y );                                  // C99
double  fmin( double  x,  double  y );                                // C99
long  double  fminl( long  double  x,  long  double  y );             // C99
```

수학 관련 함수 (3/13)

- **지수와 로그 함수 : log10**

```
#include <math.h>

// arg 의 자연 로그(기본 base는 10) 값을 반환
float  log10f( float  arg );                          // C99
double  log10( double  arg );
long  double  log10l( long  double  arg );            // C99
```

수학 관련 함수 (4/13)

- **지수와 로그 함수** : exp, log

```
#include <math.h>

// 기본 base 가 자연 로그(e)인 arg 제곱 값을 반환
float  expf( float  arg );                          // C99
double  exp( double  arg );
long  double  expl( long  double  arg );            // C99
```

```
#include <math.h>

// arg 의 자연 로그(기본 base 는 e) 값을 반환
float  logf( float  arg );                          // C99
double  log( double  arg );
long  double  logl( long  double  arg );            // C99
```

수학 관련 함수 (5/13)

- **지수와 로그 함수** : pow, sqrt

```
#include <math.h>

// base 의 exp 승 값을 계산하여 반환
float  powf( float  base,  float  exp );                          // C99
double  pow( double  base,  double  exp );
long  double  powl( long  double  base,  long  double  exp );     // C99
```

```
#include <math.h>

// arg 의 0 이 아닌 제곱근을 계산하여 반환
float  sqrtf( float  arg );                                       // C99
double  sqrt( double  arg );
long  double  sqrtl( long  double  arg );                         // C99
```

수학 관련 함수 (6/13)

- **삼각 함수** : sin, cos, tan

```
#include <math.h>

// arg 의 sine 값을 계산하여 반환
float  sinf( float  arg );                                  // C99
double  sin( double  arg );
long  double  sinl( long  double  arg );                    // C99

// arg 의 cosine 값을 계산하여 반환
float  cosf( float  arg );                                  // C99
double  cos( double  arg );
long  double  cosl( long  double  arg );                    // C99

// arg 의 tangent 값을 계산하여 반환
float  tanf( float  arg );                                  // C99
double  tan( double  arg );
long  double  tanl( long  double  arg );                    // C99
```

수학 관련 함수 (7/13)

- **삼각 함수** : asin, acos, atan

```
#include <math.h>

// arg 의 arc  sine 값을 계산하여 반환
float  asinf( float  arg );                                  // C99
double  asin( double  arg );
long  double  asinl( long  double  arg );                    // C99

// arg 의 arc  cosine 값을 계산하여 반환
float  acosf( float  arg );                                  // C99
double  acos( double  arg );
long  double  acosl( long  double  arg );                    // C99

// arg 의 arc  tangent 값을 계산하여 반환
float  atanf( float  arg );                                  // C99
double  atan( double  arg );
long  double  atanl( long  double  arg );                    // C99
```

수학 관련 함수 (8/13)

- **삼각 함수** : atan2

```
#include <math.h>

// y/x 의 arc  tangent 값을 계산하여 반환
float  atan2f( float  y,  float  x );                                  // C99
double  atan2( double  y,  double  x );                                // C99
long  double  atan2l( long  double  y,  long  double  x );             // C99
```

수학 관련 함수 (9/13)

- **쌍곡선 함수** : sinh, cosh, tanh

```
#include <math.h>

// arg 의 쌍곡선(hyperbolic)  sine 값을 계산하여 반환
float  sinhf( float  arg );                                  // C99
double  sinh( double  arg );
long  double  sinhl( long  double  arg );                    // C99

// arg 의 쌍곡선(hyperbolic)  cosine 값을 계산하여 반환
float  coshf( float  arg );                                  // C99
double  cosh( double  arg );
long  double  coshl( long  double  arg );                    // C99

// arg 의 쌍곡선(hyperbolic)  tangent 값을 계산하여 반환
float  tanhf( float  arg );                                  // C99
double  tanh( double  arg );
long  double  tanhl( long  double  arg );                    // C99
```

수학 관련 함수 (10/13)

- **쌍곡선 함수** : asinh, acosh, atanh

```
#include <math.h>

// arg 의 쌍곡선(hyperbolic)  arc  sine 값을 계산하여 반환
float  asinhf( float  arg );                                  // C99
double  asinh( double  arg );
long  double  asinhl( long  double  arg );                    // C99

// arg 의 쌍곡선(hyperbolic)  arc  cosine 값을 계산하여 반환
float  acoshf( float  arg );                                  // C99
double  acosh( double  arg );
long  double  acoshl( long  double  arg );                    // C99

// arg 의 쌍곡선(hyperbolic)  arc  tangent 값을 계산하여 반환
float  atanhf( float  arg );                                  // C99
double  atanh( double  arg );
long  double  atanhl( long  double  arg );                    // C99
```

수학 관련 함수 (11/13)

- 가장 가까운 정수 또는 부동 소수점 연산 함수 (1/2)

```
#include <math.h>

// arg 를 초과하는 정수 중에서 가장 가까운 정수를 반환
float  ceilf( float  arg );                                  // C99
double  ceil( double  arg );
long  double  ceill( long  double  arg );                    // C99

// arg 보다 작은 정수 중에서 가장 가까운 정수를 반환
float  floorf( float  arg );                                 // C99
double  floor( double  arg );
long  double  floorl( long  double  arg );                   // C99

// arg 의 소수점 이하를 버리고 정수를 반환
float  truncf( float  arg );                                 // C99
double  trunc( double  arg );                                // C99
long  double  truncl( long  double  arg );                   // C99
```

수학 관련 함수 (12/13)

- 가장 가까운 정수 또는 부동 소수점 연산 함수 (2/2)

```
#include <math.h>

// arg 에서 가까운 정수(즉, 반올림 값)를 반환
float  roundf( float  arg );                          // C99
double  round( double  arg );                         // C99
long  double  roundl( long  double  arg );            // C99

long  lroundf( float  arg );                          // C99
long  lround( double  arg );                          // C99
long  lroundl( long  double  arg );                   // C99

long  long  llroundf( float  arg );                   // C99
long  long  llround( double  arg );                   // C99
long  long  llroundl( long  double  arg );            // C99
```

수학 관련 함수 (13/13)

- **부동 소수점 조작 함수** : modf, ldexp

```
#include <math.h>

// 실수 x 를 정수 부분과 실수 부분으로 분리하여 한다.
// 정수부분은 iptr 에 저장되고, 실수부분은 반환
float  modff( float  x,  float  *iptr );                                  // C99
double  modf( double  x,  double  *iptr );
long  double  modfl(long  double  x,  long  double  *iptr);               // C99
```

```
#include <math.h>

// arg * 2^exp 를 계산하여 반환
// arg * pow(2, exp) 의 결과와 동일하다.
float  ldexpf( float  arg,  int  exp );                                   // C99
double  ldexp( double  arg,  int  exp );
long  double  ldexpl( long  double  arg,  int  xp );                      // C99
```

참고문헌

[1] 서두옥, 이동호(감수), (열혈강의)"또 하나의 C : 프로그래밍은 셀프입니다", 프리렉, 2012.

[2] Paul Deitel, Harvey Deitel, "C How to Program", Global Edition, 8/E, Pearson, 2016.

[3] SAMUEL P. HARBISON Ⅲ, GUY L. STEELE, "C 프로그래밍 언어, C : A Reference Manual", 5/E, Pearson Education Korea, 2005.

[4] 문병로, "쉽게 배우는 알고리즘 - 관계 중심의 사고법", 개정판, 한빛아카데미, 2018.

[5] 주우석, "C・C++ 로 배우는 자료구조론", 한빛아카데미, 2015.

[6] Behrouz A. Forouzan, Richard F. Gilberg, 김진 외 7인 공역, "구조적 프로그래밍 기법을 위한 C", 도서출판 인터비젼, 2004.

[7] Brian W. Kernighan, Dennis M. Ritchie, 김석환 외 2인 공역, "The C Programming Language", 2/E, 대영사, 2004.

[8] 김일광, "C 프로그래밍 입문 : 프로그래밍을 모국어처럼 유창하게", 한빛미디어, 2004.

[9] 정재은, "다시 체계적으로 배우는 C 언어 포인터", 정보문화사, 2003.