



## 이번 장에서 학습할 내용

- 반복의 개념 이해
- 변수의 속성
- 전역, 지역 변수
- 자동 변수와 정적 변수
- 재귀 호출

이번 장에서는 함수와 변수와의 관계를 집중적으로 살펴볼 것이다. 또한 함수가 자기 자신을 호출하는 재귀 호출에 대하여 살펴본다.

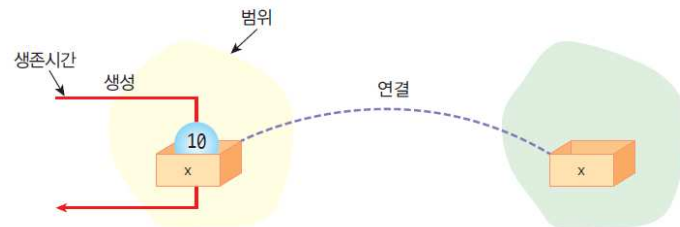


© 2012 영남대학교 All rights reserved

쉽게 풀어서 C언어 Express

## 변수의 속성

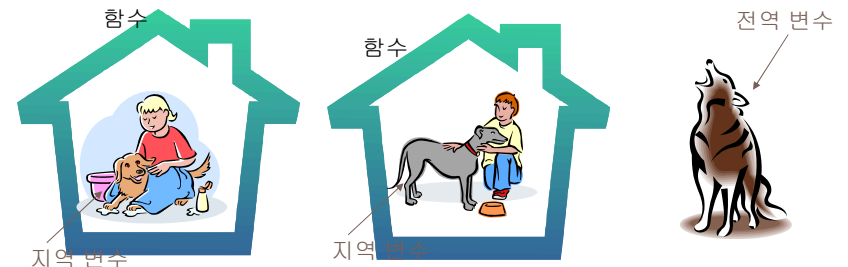
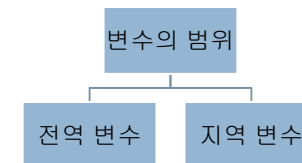
- 변수의 속성 : 이름, 타입, 크기, 값 + 범위, 생존 시간, 연결
  - 범위(scope) : 변수가 사용 가능한 범위, 가시성
  - 생존 시간(lifetime) : 메모리에 존재하는 시간
  - 연결(linkage) : 다른 영역에 있는 변수와의 연결 상태



© 2012 영남대학교 All rights reserved

쉽게 풀어서 C언어 Express 1//22(컴프 4주차)

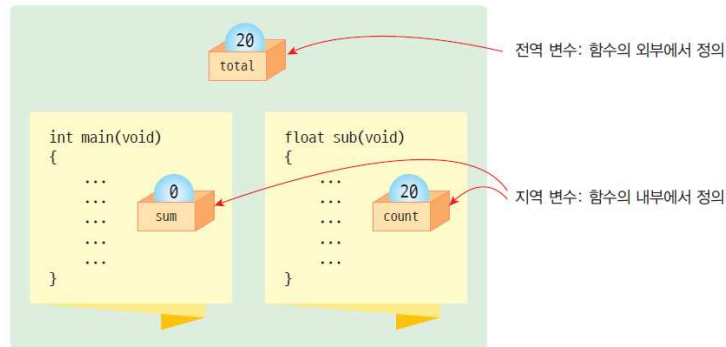
## 변수의 범위



© 2012 영남대학교 All rights reserved

쉽게 풀어서 C언어 Express

## 전역 변수와 지역 변수

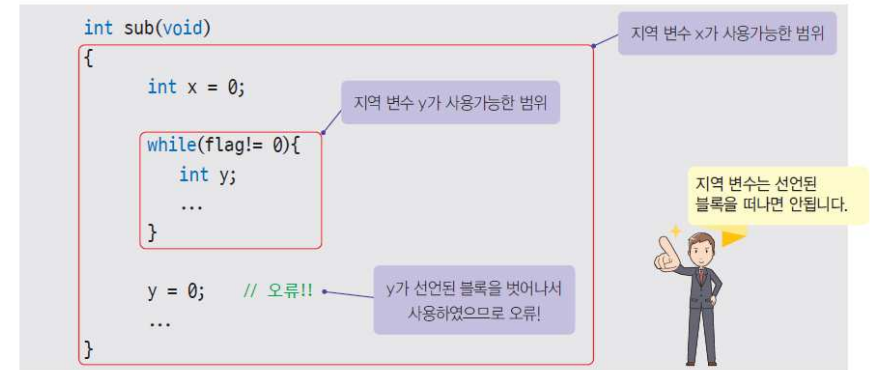


© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express

## 지역 변수

□ 지역 변수(local variable)는 블록 안에 선언되는 변수

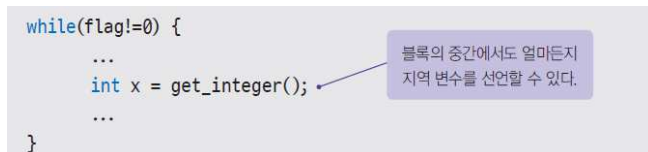


© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express

## 지역 변수 선언 위치

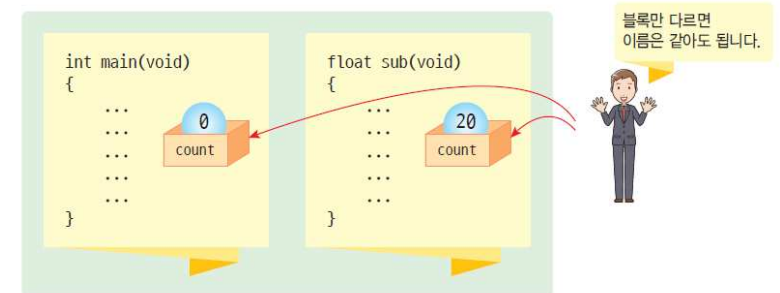
□ 최신 버전의 C에서는 블록 안의 어떤 위치에서도 선언 가능!!



© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express 2//22(컴프 4주차)

## 이름이 같은 지역 변수

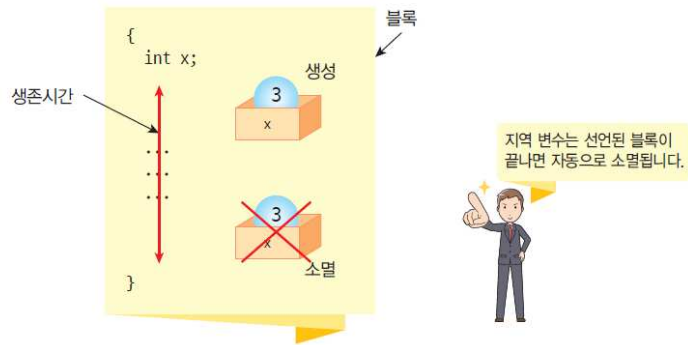


© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express



## 지역 변수의 생존 기간



© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express



## 지역 변수 예제

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

```
int main(void)
{
    int i;

    for(i = 0; i < 5; i++)
    {
        int temp = 1;
        printf("temp = %d\n", temp);
        temp++;
    }

    return 0;
}
```

블록이 시작할 때 마다 생성되어 초기화된다.

```
temp = 1
temp = 1
temp = 1
temp = 1
temp = 1
```

© 2012 영남대학교 All rights reserved

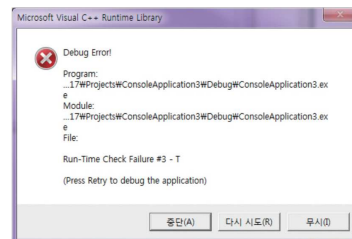
쉽게 풀어쓴 C 언어 Express



## 지역 변수의 초기값

```
#include <stdio.h>
int main(void)
{
    int temp;
    printf("temp = %d\n", temp);
}
```

초기와 되지 않았으므로 쓰레기 값은 가질다.



© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express 3//22(컴프 4주차)



## 함수의 매개 변수

```
int inc(int counter)
{
    counter++;
    return counter;
}
```

매개 변수도 일종의 지역 변수

© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express

## 함수의 매개 변수

```
#include <stdio.h>
int inc(int counter);
```

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

```
    int i;
```

```
    i = 10;
```

```
    printf("함수 호출전 i=%d\n", i);
```

```
    inc(i);
```

```
    printf("함수 호출후 i=%d\n", i);
```

```
    return 0;
```

```
}
```

```
void inc(int counter)
```

```
{
```

```
    counter++;
```

```
}
```

값에 의한 호출  
(call by value)

매개 변수도 일종의  
지역 변수임

함수 호출전 i=10

함수 호출후 i=10

© 2

쉽게 풀어쓴 C 언어 Express

## 전역 변수

- 전역 변수(global variable)는 함수 외부에서 선언되는 변수이다.
- 전역 변수의 범위는 소스 파일 전체이다.

© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express

## 전역 변수의 초기값과 생성 시간

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

```
int A;
```

```
int B;
```

```
int add()
```

```
{
```

```
    return A + B;
```

```
}
```

```
int main()
```

```
{
```

```
    int answer;
```

```
    A = 5;
```

```
    B = 7;
```

```
    answer = add();
```

```
    printf("%d + %d = %d\n", A, B, answer);
```

```
    return 0;
```

```
}
```

전역 변수  
초기값은 0

5 + 7 = 12

© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express 4//22(컴프 4주차)

## 전역 변수의 초기값

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

```
int counter;
```

```
int main(void)
```

```
{
```

```
    printf("counter=%d\n", counter);
```

```
    return 0;
```

```
}
```

counter=0

© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express

## 전역 변수의 사용

```
#include <stdio.h>
int x;
void sub();

int main(void)
{
    for(x=0; x<10; x++)
        sub();
}

void sub()
{
    for(x=0; x<10; x++)
        printf("%d\n", x);
}
```



©

쉽게 풀어쓴 C언어 Express

## 전역 변수의 사용

- 거의 모든 함수에서 사용하는 공통적인 데이터는 전역 변수로 한다.
- 일부의 함수들만 사용하는 데이터는 전역 변수로 하지 말고 함수의 인수로 전달한다.

© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C언어 Express

## 같은 이름의 전역 변수와 지역 변수

```
#include <stdio.h>

int sum = 1; // 전역 변수

int main(void)
{
    int sum = 0; // 지역 변수

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

    return 0;
}
```

전역 변수와 지역 변수가 동일한 이름으로 선언된다.

sum = 0

© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C언어 Express 5//22(컴프 4주차)

## 중간 점검

- 변수의 범위는 대개 무엇으로 결정되는가?
- 변수의 범위에는 몇 가지의 종류가 있는가?
- 지역 변수를 블록의 중간에서 정의할 수 있는가?
- 지역 변수가 선언된 블록이 종료되면 지역 변수는 어떻게 되는가?
- 지역 변수의 초기값은 얼마인가?
- 전역 변수는 어디에 선언되는가?
- 전역 변수의 생존 기간과 초기값은?
- 똑같은 이름의 전역 변수와 지역 변수가 동시에 존재하면 어떻게 되는가?



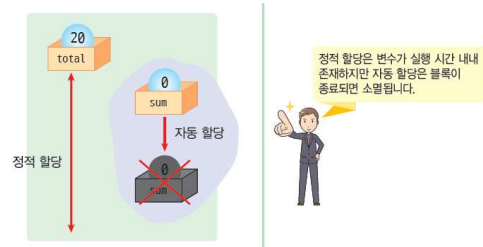
© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C언어 Express



## 생존 기간

- 정적 할당(static allocation):
  - ▣ 프로그램 실행 시간 동안 계속 유지
- 자동 할당(automatic allocation):
  - ▣ 블록에 들어갈 때 생성
  - ▣ 블록에서 나올 때 소멸



© 2012 컴퓨팅학 All rights reserved

쉽게 풀어쓴 C 언어 Express



## 생존 기간

- 생존 기간을 결정하는 요인
  - ▣ 변수가 선언된 위치
  - ▣ 저장 유형 지정자
- 저장 유형 지정자
  - ▣ auto
  - ▣ register
  - ▣ static
  - ▣ extern

© 2012 컴퓨팅학 All rights reserved

쉽게 풀어쓴 C 언어 Express



## 저장 유형 지정자 auto

- 변수를 선언한 위치에서 자동으로 만들어지고 블록을 벗어나게 되며 자동으로 소멸되는 저장 유형을 지정
- 지역 변수는 **auto**가 생략되어도 자동 변수가 된다.

```
int main(void)
{
    auto int sum = 0;
    int i = 0;
    ...
    ...
}
```

전부 자동 변수로서 함수가 시작되면 생성되고 끝나면 소멸된다.

© 2012 컴퓨팅학 All rights reserved

쉽게 풀어쓴 C 언어 Express 6//22(컴프 4주차)



## 저장 유형 지정자 static

```
#include <stdio.h>

void sub() {
    static int scount = 0;
    int account = 0;
    printf("scount = %d\t", scount);

    printf("account = %d\n", account);
    scount++;
    account++;
}

int main(void) {
    sub();
    sub();
    sub();
    return 0;
}
```

scount = 0 account = 0  
scount = 1 account = 0  
scount = 2 account = 0

정적 지역 변수로서 static을 붙이면 지역 변수가 정적 변수로 된다,

© 2012 컴퓨팅학 All rights reserved

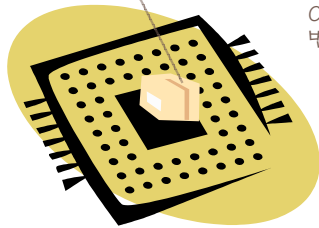
쉽게 풀어쓴 C 언어 Express



## 저장 유형 지정자 register

- 레지스터(register)에 변수를 저장.

```
register int i;
for(i = 0; i < 100; i++)
    sum += i;
```



CPU안의 레지스터에  
변수가 저장됨



## volatile

- volatile 지정자는 하드웨어가 수시로 변수의 값을 변경하는 경우에 사용된다

```
volatile int io_port; // 하드웨어와 연결된 변수
```

```
void wait(void) {
    io_port = 0;
    while (io_port != 255)
        ;
}
```



## 중간 점검

- 저장 유형 지정자에는 어떤 것들이 있는가?
- 지역 변수를 정적 변수로 만들려면 어떤 지정자를 붙여야 하는가?
- 변수를 CPU 내부의 레지스터에 저장시키는 지정자는?
- 컴파일러에게 변수가 외부에 선언되어 있다고 알리는 지정자는?
- static 지정자를 변수 앞에 붙이면 무엇을 의미하는가?



## lab: 은행 계좌 구현하기

- 돈만 생기면 저금하는 사람을 가정하자. 이 사람을 위한 함수 **save(int amount)**를 작성하여 보자. 이 함수는 저금할 금액을 나타내는 인수 **amount**만을 받으며 **save(100)**과 같이 호출된다. **save()**는 정적 변수를 사용하여 현재까지 저축된 총액을 기억하고 있으며 한번 호출될 때마다 총 저축액을 다음과 같이 화면에 출력한다.





while(1)

사용자로부터 아이디를 입력받는다.

사용자로부터 패스워드를 입력받는다.

만약 로그인 시도 횟수가 일정 한도를 넘었으면 프로그램을 종료한다.

아이디와 패스워드가 일치하면 로그인 성공 메시지를 출력한다.

아이디와 패스워드가 일치하지 않으면 다시 시도한다.



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

```
// amount가 양수이면 입금이고 음수이면 출금으로 생각한다.
```

```
void save(int amount)
```

```
{
```

```
    static long balance = 0;
```

```
    if( amount >= 0)
```

```
        printf("%d \t\t", amount);
```

```
    else
```

```
        printf("\t %d \t", -amount);
```

```
    balance += amount;
```

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

```
}
```

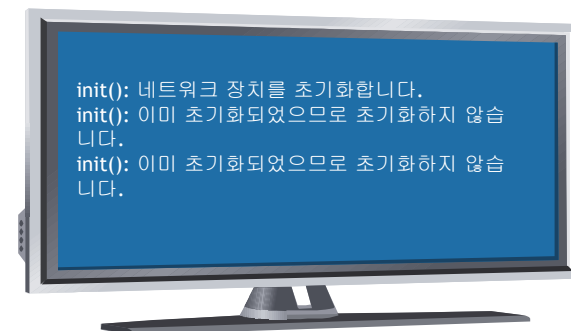


```
int main(void) {  
    printf("=====================\n");  
    printf("입금 \t\t출금 \t\t\n");  
    printf("=====================\n");  
    save(10000);  
    save(50000);  
    save(-10000);  
    save(30000);  
    printf("=====================\n");  
    return 0;  
}
```



## lab: 한번만 초기화하기

- 정적 변수는 한번만 초기화하고 싶은 경우에도 사용된다







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

void init();

int main(void)
{
    init();
    init();
    init();
    return 0;
}

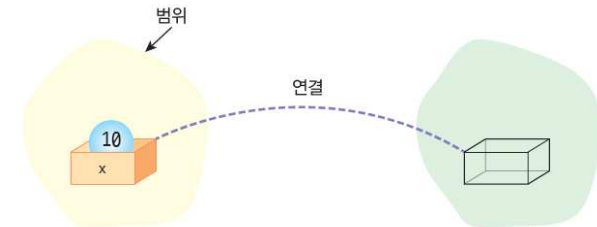
void init()
{
    static int initd = 0;
    if( initd == 0 ){
        printf("init(): 네트워크 장치를 초기화합니다. \n");
        initd = 1;
    }
    else {
        printf("init(): 이미 초기화되었으므로 초기화하지 않습니다. \n");
    }
}
```

쉽게 풀어쓴 C 언어 Express



## 연결

- 연결 (*linkage*): 다른 범위에 속하는 변수들을 서로 연결하는 것
  - 외부 연결
  - 내부 연결
  - 무연결
- 전역 변수만이 연결을 가질 수 있다.



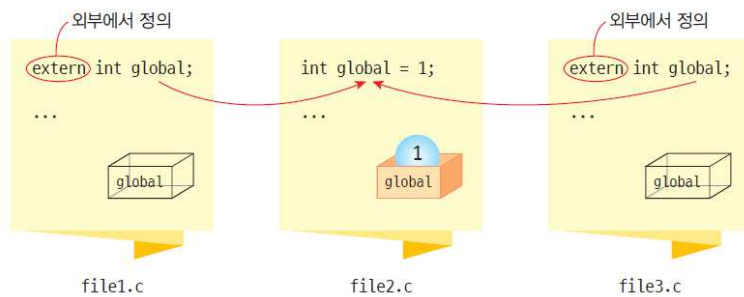
© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express



## 외부 연결

- 전역 변수를 **extern**을 이용하여서 서로 연결



© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express 9/22(컴프 4주차)



## 연결 예제

```
linkage1.c
#include <stdio.h>
int all_files; // 다른 소스 파일에서도 사용할 수 있는 전역 변수
static int this_file; // 현재의 소스 파일에서만 사용할 수 있는 전역 변수
extern void sub();

int main(void)
{
    sub();
    printf("%d\n", all_files);
    return 0;
}

linkage2.c
extern int all_files;
void sub(void)
{
    all_files = 10;
}

10
```

© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C 언어 Express

## 함수의 static

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

extern void f2();
int main(void)
{
    f2();
    return 0;
}

sub.c
static void f1()
{
    printf("f1()이 호출되었습니다.\n");
}
void f2()
{
    f1();
    printf("f2()가 호출되었습니다.\n");
}
```

Static이 붙는 함수는 파일 안에서만 사용할 수 있다.

© 2012 '헛소리' All rights reserved

쉽게 풀어쓴 C 언어 Express

## 저장 유형 정리

- 일반적으로는 *자동 저장 유형* 사용 권장
- 변수의 값이 함수 호출이 끝나도 그 값을 유지하여야 할 필요가 있다면 *지역 정적*
- 만약 많은 함수에서 공유되어야 하는 변수라면 *외부 참조 변수*

저장 유형	키워드	정의되는 위치	범위	생존 시간
자동	auto	함수 내부	지역	임시
레지스터	register	함수 내부	지역	임시
정적 지역	static	함수 내부	지역	영구
전역	없음	함수 외부	모든 소스 파일	영구
정적 전역	static	함수 외부	하나의 소스 파일	영구
외부 참조	extern	함수 외부	모든 소스 파일	영구

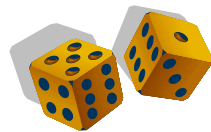
© 2012 '헛소리' All rights reserved

쉽게 풀어쓴 C 언어 Express

## 예제: 난수 발생기

- 자체적인 난수 발생기 작성
- 이전에 만들어졌던 난수를 이용하여 새로운 난수를 생성함을 알 수 있다. 따라서 함수 호출이 종료되더라도 이전에 만들어졌던 난수를 어딘가에 저장하고 있어야 한다

$$r_{n+1} = (a \cdot r_n + b) \bmod M$$



© 2012 '헛소리' All rights reserved

쉽게 풀어쓴 C 언어 Express

## 예제

```
main.c
#include <stdio.h>
unsigned random_i(void);
double random_f(void);

extern unsigned call_count; // 외부 참조 변수

int main(void)
{
    register int i; // 레지스터 변수

    for(i = 0; i < 10; i++)
        printf("%d ", random_i());

    printf("\n");

    for(i = 0; i < 10; i++)
        printf("%f ", random_f());

    printf("\n함수가 호출된 횟수= %d\n", call_count);
    return 0;
}
```

© 2012 '헛소리' All rights reserved

쉽게 풀어쓴 C 언어 Express



## 예제

random.c

```
// 난수 발생 함수
#define SEED 17
#define MULT 25173
#define INC 13849
#define MOD 65536
```

```
48574 61999 40372 31453 39802 35227 15504 29161
14966 52039
0.885437 0.317215 0.463654 0.762497 0.546997
0.768570 0.422577 0.739731 0.455627 0.720901
함수가 호출된 횟수 = 20
```

```
unsigned int call_count = 0; // 전역 변수
static unsigned seed = SEED; // 정적 전역 변수
```

```
unsigned random_i(void)
{
    seed = (MULT*seed + INC) % MOD;
    call_count++;
    return seed;
}
double random_f(void)
{
    seed = (MULT*seed + INC) % MOD;
    call_count++;
    return seed / (double) MOD;
}
```

© 2012



## 가변 매개 변수

- 매개 변수의 개수가 가변적으로 변할 수 있는 기능

호출 때 매개 변수의 개수가  
변경될 수 있다.

```
int sum( int num, ... )
```

© 2012 컴플렉스 All rights reserved

쉽게 풀어쓴 C언어 Express



## 가변 매개 변수

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

합은 10입니다.

```
{
    int answer = sum( 4, 4, 3, 2, 1 );
    printf( "합은 %d입니다.\n", answer );
    return( 0 );
}

int sum( int num, ... )
{
    int answer = 0;
    va_list argptr;
    va_start( argptr, num );
    for( ; num > 0; num-- )
        answer += va_arg( argptr, int );
    va_end( argptr );
    return( answer );
}
```

11/22(컴프 4주차)



## 순환(recursion)이란?

- 함수는 자기 자신을 호출할 수도 있다. 이것을 순환 (recursion)라고 부른다.

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

© 2012 컴플렉스 All rights reserved

쉽게 풀어쓴 C언어 Express



## 팩토리얼 구하기

- 팩토리얼 프로그래밍:  $(n-1)!$  팩토리얼을 현재 작성중인 함수를 다시 호출하여 계산(순환 호출)

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



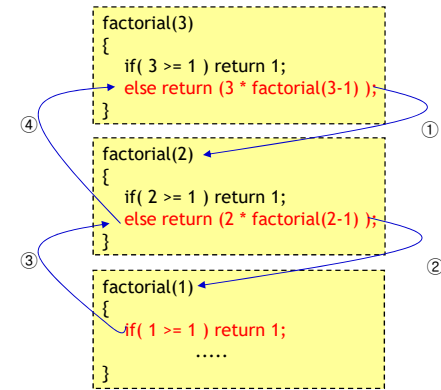
쉽게 풀어쓴 C언어 Express



## 팩토리얼 구하기

- 팩토리얼의 호출 순서

```
factorial(3) = 3 * factorial(2)
              = 3 * 2 * factorial(1)
              = 3 * 2 * 1
              = 3 * 2
              = 6
```



© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C언어 Express



## 팩토리얼 계산

```
// 재귀적인 팩토리얼 함수 계산
#include <stdio.h>

long factorial(int n)
{
    printf("factorial(%d)\n", n);

    if(n <= 1) return 1;
    else return n * factorial(n - 1);
}

int main(void)
{
    int x = 0;
    long f;

    printf("정수를 입력하십시오:");
    scanf("%d", &n);
    printf("%d!은 %d입니다. \n", n, factorial(n));
    return 0;
}
```

© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C언어 Express 12/22(컴프 4주차)



## 2진수 형식으로 출력하기

```
// 2진수 형식으로 출력
#include <stdio.h>

void print_binary(int x);

int main(void)
{
    print_binary(9);
}

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

© 2012 영남대학교 All rights reserved

쉽게 풀어쓴 C언어 Express



## 최대 공약수 구하기

```
// 최대 공약수 구하기
#include <stdio.h>

int gcd(int x, int y);

int main(void)
{
    printf("%d\n", gcd(30, 20));
}

// x는 y보다 커야 한다.
int gcd(int x, int y)
{
    if (y == 0)
        return x;
    else
        return gcd(y, x % y);
}
```

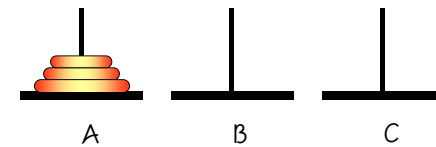
© 2012 썬데이컴퓨터 All rights reserved

쉽게 풀어쓴 C언어 Express



## 하노이 탑 문제 #1

- 문제는 막대 A에 쌓여있는 원판 3개를 막대 C로 옮기는 것이다. 단 다음의 조건을 지켜야 한다.
  - 한 번에 하나의 원판만 이동할 수 있다
  - 맨 위에 있는 원판만 이동할 수 있다
  - 크기가 작은 원판 위에 큰 원판이 쌓일 수 없다.
  - 중간의 막대를 임시적으로 이용할 수 있으나 앞의 조건들을 지켜야 한다.

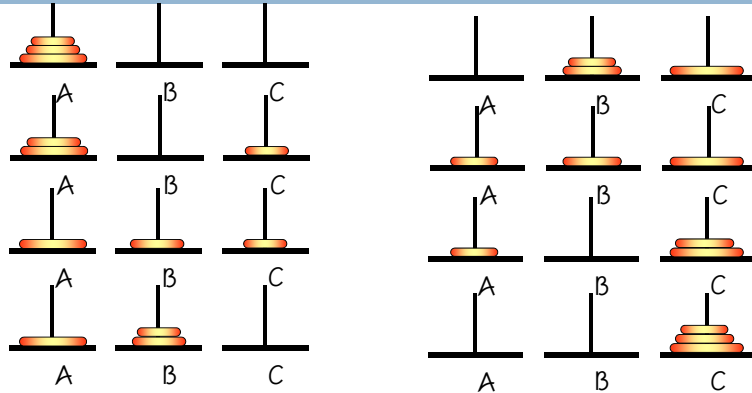


© 2012 썬데이컴퓨터 All rights reserved

쉽게 풀어쓴 C언어 Express



## 3개의 원판인 경우의 해답



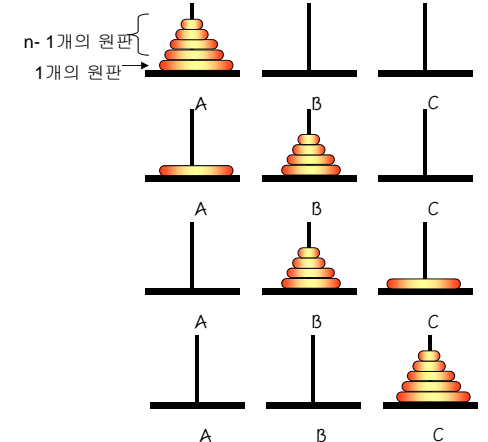
© 2012 썬데이컴퓨터 All rights reserved

쉽게 풀어쓴 C언어 Express 13//22(컴프 4주차)



## n개의 원판인 경우

- n-1개의 원판을 A에서 B로 옮기고 n번째 원판을 A에서 C로 옮긴 다음, n-1개의 원판을 B에서 C로 옮기면 된다.



© 2012 썬데이컴퓨터 All rights reserved

쉽게 풀어쓴 C언어 Express



## 하노이탑 알고리즘

// 막대 from에 쌓여있는 n개의 원판을 막대 tmp를 사용하여 막대 to로 옮긴다.

```
void hanoi_tower(int n, char from, char tmp, char to)
{
    if (n == 1)
    {
        from에서 to로 원판을 옮긴다.
    }
    else
    {
        hanoi_tower(n-1, from, to, tmp);
        from에 있는 한 개의 원판을 to로 옮긴다.
        hanoi_tower(n-1, tmp, from, to);
    }
}
```

© 2012 컴퓨팅학 All rights reserved

쉽게 풀어쓴 C언어 Express



## 하노이탑 실행 결과

```
#include <stdio.h>

void hanoi_tower(int n, char from, char tmp, char to)
{
    if (n == 1)
        printf("원판 1을 %c 에서 %c로 옮긴다.\n", from, to);
    else {
        hanoi_tower(n-1, from, to, tmp);
        printf("원판 %d를 %c에서 %c로 옮긴다.\n", n, from, to);
        hanoi_tower(n-1, tmp, from, to);
    }
}

int main(void)
{
    hanoi_tower(4, 'A', 'B', 'C');
    return 0;
}
```

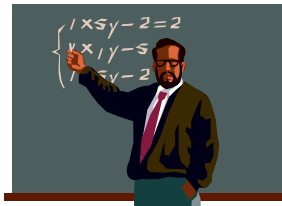
원판 1을 A 에서 B로 옮긴다.  
원판 2을 A에서 C으로 옮긴다.  
원판 1을 B 에서 C으로 옮긴다.  
원판 3을 A에서 B으로 옮긴다.  
원판 1을 C 에서 A으로 옮긴다.  
원판 2을 C에서 B으로 옮긴다.  
원판 1을 A 에서 B으로 옮긴다.  
원판 4을 A에서 C으로 옮긴다.  
원판 1을 B 에서 C으로 옮긴다.  
원판 2을 B에서 A으로 옮긴다.  
원판 1을 C 에서 A으로 옮긴다.  
원판 3을 B에서 C으로 옮긴다.  
원판 1을 A 에서 B으로 옮긴다.  
원판 2을 A에서 C으로 옮긴다.  
원판 1을 B 에서 C으로 옮긴다.

© 2012 컴퓨팅학 All rights reserved

쉽게 풀어쓴 C언어 Express



## Q & A



© 2012 컴퓨팅학 All rights reserved

쉽게 풀어쓴 C언어 Express

14//22(컴프 4주차)

컴프 수업 #1. 함수 revisited: 순환(재귀, recursion)

**순환(recursion):** 알고리즘이나 함수가 수행 도중에 자기 자신을 다시 호출하여 문제를 해결하는 기법  
이러한 함수를 **재귀함수**라고도 부른다.

## ■ 반복 vs 순환

주어진  $n$ 에 대해서  $n!$ 를 계산하여 반환하는 함수를 생각해 보자.

<pre>// 반복적 방법 int factorial(int n) {     // ... } </pre>	<pre>// 재귀적(순환적) 방법 int factorial(int n) {     // ... } </pre>
---	--

팩토리알 계산의 경우 다음과 같이 재귀적으로 정의가 가능하다.

$$f(0) = 0! = 1$$
$$f(n) = n! = n * (n - 1)! = n * f(n - 1)$$

이런 경우 재귀함수를 사용하여 정의가 가능하다.

어떤 경우 재귀적 방법으로 변환 가능한가?

- 재귀 함수에서는  
재귀 호출하는 부분과 순환호출을 멈추는 부분이 반드시 포함되어야한다.

## ■ 연습

다음은 재귀적으로 정의가 가능한가? 정의식은?

$$- 1 + 2 + \dots + n$$

$$- 1 + 1/2 + 1/3 + \dots + 1/n$$

$$- 2^n$$

$$-x^y$$

- 피보나치 수열(1 1 2 3 5 8 13 21 34 ...)

## LAB 4 함수 고급

### ■ LAB4\_1 (전역변수 연습 + 여러 개의 소스파일)

□ LAB4\_1\_1 아래는 간단한 샘플 프로그램이다. 아래 프로그램을 빌드 한 후에 아래 지시에 따른다.

/\* 초기 프로그램, 1개 파일로 프로그램 작성 \*/

```
#include <stdio.h>
float g_i = 10.0;

float sum(float x, float y)
{
    return x + y;
}

float sum_2(float a, float b)
{
    return g_i + sum(a, b);
}

int main(void)
{
    fprintf(stdout, "계산 결과는 %f 입니다\n", sum_2(10, 20));
    return 0;
}
```

전 주에 배운 F11 키와 F10을 적절히 사용하여 프로그램의 모든 라인을 한 라인씩 수행해 보자. 이를 통해 함수가 수행되는 순서와 각 함수가 호출 될 때의 매개변수 값을 적어본다. (VS의 아래창에 변수값들이 보인다)

□ LAB4\_1\_2 아래와 같이 두 개의 소스 파일로 위의 프로그램을 분리해 보자. 소스파일의 이름은 main.c, function.c로 한다. 그리고 빌드해 보자. 어떤 컴파일 에러 및 링크 에러가 출력되는지 기록하고 이를 해석해 보자. (새로운 파일을 프로젝트에 추가하지 못하는 학생은 교수님 지도에 따를 것)

main.c

```
#include <stdio.h>
float g_i = 10.0;

int main(void)
{
    fprintf(stdout, "계산결과는 %f.\n", sum_2(10, 20));
    return 0;
}
```

function.c

```
float sum_2(float a, float b)
{
    return g_i + sum(a, b);
}

float sum(float a, float b)
{
    return a + b;
}
```

가) 위의 두 개 파일로 구성된 프로그램이 올바르게 빌드되도록 프로그램을 수정하여 보시오. (힌트: 함수의 경우 호출하려는 함수가 외부 파일에 정의되어 있을 때도 함수 프로토타입을 적어 주면 VS가 링크 단계에서 해당 함수를 찾아 올바르게 빌드함. 전역 변수의 경우는 배운 내용에 따르시오)

나) 빌드가 되었다면 나)번에서 적어둔 빌드시의 에러 메시지를 확인하여 그 의미를 다시 한번 복습하시오. 그리고, 가)번과 동일하게 모든 라인을 디버거로 수행해 보시오.

■ LAB4\_2(재귀 함수 연습) 원편에 주어진 sum(int n)은 1부터 n까지의 합을 구해 반환해 주는 함수이다. 이와 같은 일을 하는 함수 sum\_rec(int n)을 재귀함수를 이용하여 작성하고자 한다. 아래 힌트와 주어진 코드 일부를 사용하여 sum\_rec() 함수를 완성해 보시오.

(힌트: sum(n) = n + sum(n - 1))

```
#include <stdio.h>
int sum(int);
void main()
{
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    printf("1부터 %d까지의 합은 %d입니다,\n",
        n, sum(n));
}

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

```
#include <stdio.h>
int sum_rec(int);
void main()
{
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    printf("1부터 %d까지의 합은 %d입니다,\n",
        n, sum_rec(n));
}

// 재귀함수 이용
int sum_rec(int n)
{
    if ( ????? ) // 이 곳 코드 삽입
        return ____;
    else
        return ____;
}
```

가) 빌드가 되었다면 디버거 기능으로 한 라인씩 수행시켜 보자. 이 때 sum\_rec() 함수가 수행될 때 마다 매개변수 n의 값이 어떻게 변화하는지 순서대로 적어본다.

나) 가)번의 내용을 바탕으로 위의 함수 sum\_rec()이 어떻게 1부터 n까지의 합을 구하고 있는지 다시 한번 생각해 본다.

다) 위의 sum()과 sum\_rec() 중 어느것이 코딩하기 쉬운가? 또, 둘 중 어느것이 이해하기 쉬운가?



■ **LAB4.3** 피보나치 수열의 값을 출력하려한다. 지시된 내용에 따라 작업을 수행하시오.

피보나치 수열은 다음과 같은 재귀적 정의에 의해 정해진다:

$F(n) = F(n-1) + F(n-2).$   
 $F(0) = F(1) = 1$

```
#include <stdio.h>
int fibo(int n);
int main(void)
{
    int n, idx;

    printf("몇개의 피보나치 수열값을 출력할까요?(3보다 큰 정수):");
    scanf("%d", &n);

    for (idx = 0; idx < n; idx++)
    {
        printf("%10d ", fibo(idx));
        if ((idx + 1) % 5 == 0)    // 한줄에 5개씩 출력하기위해서..
            printf("\n");
    }
    printf("\n");

    int fibo(int n)
    {
        if (n == 0 || n == 1)
            /* 이곳에 코드 삽입 */

        else
            /* 이곳에 코드 삽입 */

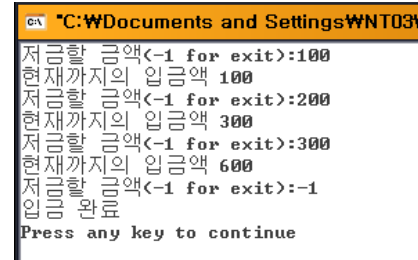
    }
```

**잔소리)** 여러분이 프로그램을 잘 짜기 위해서는 디버거를 사용하여 프로그램을 자세히 분석하는 습관(한 줄씩 수행하고 변수가 어떻게 변화하는지)을 가져야 합니다. 이 때, 자신이 짠 프로그램 외에 교재의 연습문제와 같이 자신보다 실력이 좋은 사람이 짠 프로그램을 사용하는 것이 좋습니다.

## HW 4

■ **HW4.1(static 변수의 사용)**(난이도 중하)(1학기 내용 복습) 아래와 같이 입금만을 처리하는 프로그램을 완성하라.main함수외에 아래와 같은 prototype을 갖는 함수를 사용하라.

```
void save(int money);
```



```
C:\WDocuments and Settings\WNT03\
저금할 금액<-1 for exit>:100
현재까지의 입금액 100
저금할 금액<-1 for exit>:200
현재까지의 입금액 300
저금할 금액<-1 for exit>:300
현재까지의 입금액 600
저금할 금액<-1 for exit>:-1
입금 완료
Press any key to continue
```

- static 변수를 사용하라.
- 1학기에 배운 감시값 제어반복문의 구조를 이용하라.

■ **HW4.2(난이도 중상)(Challenge)**  
재귀 함수를 사용하지 않고 다시 프로그램하라.  
즉 main함수는 그대로 두고 함수 fibo만 변경하라.

- 체크 포인트
- 입력을 40으로 하고 실행시켜보라. lab4\_3과 hw4\_1의 속도를 비교해보라. 어느것이 빠른가?
  - LAB4\_3과 HW4\_2중 어느것이 프로그램하기 더 쉬운가?

```
#include <stdio.h>
int fibo(int n);
int main(void)
{
    int n, idx;

    printf("몇개의 피보나치 수열값을 출력할까요?(3보다 큰 정수):");
    scanf("%d", &n);

    for (idx = 0; idx < n; idx++)
    {
        printf("%d ", fibo(idx));
        if ((idx + 1) % 5 == 0)
            printf("\n");
    }
    printf("\n");

    int fibo(int n) // 재귀적으로 구현하지 않는다
    {

    }
```

■ HW4\_3(난이도 중하 - 재귀함수를 이해했다면)

□ HW4\_3.0

2의 n승을 구하는 함수를 재귀적으로 구현해 보자. 주어진 main은 그대로 사용한다.

```
#include <stdio.h>

int twoPower(int x);
int main(void)
{
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);

    printf("2의 %d승은 %d이다\n", n, twoPower(n));
}

int twoPower(int x)
{
    if (x == 0)
        /* 이곳에 적절한 return문을 넣는다.
    Else
        /* 이곳에 적절한 return문을 넣는다.
}
```

□ HW4\_3.1(이름 제출한다)

x의 y승을 구하는 함수를 재귀적으로 구현해보자. 주어진 main은 그대로 사용한다.

```
#include <stdio.h>

int xPoswer(int x, int y);
int main(void)
{
    int x, y;
    printf("Enter two numbers: ");
    scanf("%d %d", &x, &y);

    printf("%d의 %d승은 %d이다\n", x, y, xPower(x, y));
}

int xPower(int x, int y)
{
    if (.....)
        /* 이곳에 적절한 return문을 넣는다.
    else
        /* 이곳에 적절한 return문을 넣는다.
}
```

■ HW4\_4 (조합의 수-순환)

수학에서 조합은 n 개에서 r 개를 뽑는 가지 수이다. 조합은 아래와 같이 재귀적으로 정의한다. 조합의 가지 수를 구하는 함수를 comb 라고 하자. 아래의 정의를 보고 comb 함수를 recursion 을 이용하여 완성하시오.

$${}_nC_r = 1 \text{ (if } r = 0 \text{ or } r = n) \\ = {}_{n-1}C_{r-1} + {}_{n-1}C_r$$

```
#include<stdio.h>
int comb(int n, int r)
{
    ...
}
int main(void)
{
    int n, r;

    printf("Enter n and r:");
    scanf("%d %d", &n, &r);
    pprintf("%d", comb(n, r));
    return 0;
}
```

■ HW4\_5(사이클 숫자)

- HW4\_5\_1(반복) 어떤 정수 n이 짝수면 2로 나누고 홀수면 3을 곱한 다음 1을 더한다. 이렇게 해서 새로 만들어진 숫자를 n으로 놓고 n=1이 될 때까지 같은 작업을 반복한다. 예를 들어 n=22이면 다음과 같은 수열이 만들어진다.

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

이 때 1이 나올 때까지 만들어진 수의 개수를 (n과 1포함) 사이클 길이라 한다. 예를 들어 n=22인 경우에는 사이클 길이가 16이다 (15아님). 특정한 수를 파라미터로 받아 위와 같은 수열을 출력하고 사이클 길이를 리턴해주는 함수를 반복문을 이용하여 작성하시오. 함수의 원형은 int get\_cycle\_number(int n); 이다.

main함수를 만들어 위의 함수를 테스트하라.

□ HW4\_5\_2(재귀)

HW4\_5\_1에서의 get\_cycle\_number(int n) 함수를 순환을 이용하여 작성하고 main함수로 테스트하라.  
- get\_cycle\_number 함수 안에서 지역변수를 사용하지 말라.

■ **Challenge 4.1** 두 정수의 최대공약수를 구하는 함수를 반복문을 이용하여 작성하시오.

- 1)
- 2) 유클리드 호제법을 이용한다!
- 3) a를 b로 나누었을 때 몫이 q이고 나머지가 r이면, a와 b의 최대공약수는 b와 r의 최대공약수와 같다.
- 4) R이 0인 경우 최대공약수는 b이다.
- 5)
- 6) 즉, a=27, b=15라 할 때,
- 7)  $27 \% 15 = 12$
- 8)  $15 \% 12 = 3$
- 9)  $12 \% 3 = 0$  // 그러므로 3이 최대 공약수이다.
- 10)

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

```
int gcd(int x, int y);
int main(void)
{
    int a, b, big, small;

    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);

    if (a < b)
    {
        big = b;
        small = a;
    }
    else
    {
        big = a;
        small = b;
    }

    printf("%d와 %d의 최대공약수는 %d\n", a, b, gcd(big, small));
}
```

```
int gcd(int x, int y)
{
```

```
}
■ Challenge 4.2 두정수의 최대공약수를 구하는 함수를 재귀함수를 이용하여 작성하시오. (위의 Challenge 4.1에서 함수 gcd 만 변경하면 된다)
```

```
int gcd(int x, int y)
{
    // 힌트: 큰수 x와 작은수 y를 매개변수로 받아서
    // y가 0이면 (탈출조건) x를 반환하고,
    // 0이 아니면 작은 수 y와 (큰수 % 작은수)를 매개변수로 자기 자신 함수를 재귀호출한다.
```

```
}
```

## 프로젝트(집합) - PREP

### ■ (PROJECT1(집합)을 위한 준비)

아래의 두 함수 isSetEqual 과 addOneElement, printSet 을 정의하여 아래처럼 실행결과가 나오게 하라.

힌트: isSetEqual 과 addOneElement 를 정의할 때 주어진 hasElement 를 사용하라.

main 함수는 그대로 둔다.

```
#define MAX_SET_SIZE 10
```

```
#define HAVE_ELEMENT 1
```

```
#define DO_NOT_HAVE_ELEMENT 0
```

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

```
// set 에 element 가 있으면 1 을 없으면 0 을 반환한다
```

```
int hasElement(int set[], int size, int element)
```

```
{
```

```
    int i;
```

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

```
        if( set[i] == element )
```

```
            return HAVE_ELEMENT; // we found it
```

```
    return DO_NOT_HAVE_ELEMENT;
```

```
}
```

```
void printSet(int set[], int size)
```

```
{...
```

```
}
```

```
// set1 과 set2 가 같으면 1 을 다르면 0 을 반환
```

```
int isSetEqual(int set1[], int size1, int set2[], int size2)
```

```
{...
```

```
}
```

```
// 원소가 집합에 존재하지 않으면 추가, 이미 존재하면 redundant 라고 출력하고 현재 집합 크기를 반환
```

```
int addOneElement(int set[], int size, int element)
```

```
{ ...
```

```
}
```

```
int main(void)
```

```
{
```

```
    int setA[MAX_SET_SIZE] = {1, 2, 3};
```

```
    int setB[MAX_SET_SIZE] = {3, 2, 1, 4};
```

```
    int num;
```

```
    int sizeA = 3, sizeB = 4;
```

```
    printf("A:"); printSet(setA, sizeA);
```

```
    printf("B:"); printSet(setB, sizeB);
```

```
    if (isSetEqual(setA, sizeA, setB, sizeB))
```

```
        printf("집합 A 와 B 는 같다\n");
```

```
    else
```

```
        printf("집합 A 와 B 는 다르다\n\n");
```

```
    printf("A 에 3 을 추가하면\n");
```

```
    sizeA = addOneElement(setA, sizeA, 3); // 3 을 SetA 에 추가한다
```

```
    printf("집합 A:"); printSet(setA, sizeA);
```

```
    printf("A 에 4 를 추가하면\n");
```

```
    sizeA = addOneElement(setA, sizeA, 4); // 4 를 SetA 에 추가한다
```

```
    printf("집합 A:"); printSet(setA, sizeA);
```

```
    if (isSetEqual(setA, sizeA, setB, sizeB))
```

```
        printf("집합 A 와 B 는 같다\n");
```

```
    else
```

```
        printf("집합 A 와 B 는 다르다\n");
```

```
}
```

```
C:\windows\system32\cmd.exe
```

```
A:< 1, 2, 3 >
```

```
B:< 3, 2, 1, 4 >
```

```
집합 A와 B는 다르다
```

```
A에 3을 추가하면
```

```
It is redundant. Please retry.
```

```
집합 A:< 1, 2, 3 >
```

```
A에 4를 추가하면
```

```
집합 A:< 1, 2, 3, 4 >
```

```
집합 A와 B는 같다
```

```
계속하려면 아무 키나 누르십시오
```

## PROJECT 01(집합) : 배열의 함수 매개변수

### ■ Project1(배열과 함수)

본 프로젝트는 여러분들이 고등학교 수학에서 가장 먼저 배우는 집합에 관한 것이다. 집합에 대한 정보는 구글이나 네이버의 검색을 통해 충분을 얻을 수 있다. Wikipedia에도 자세한 설명이 있다.  
우리가 작성하길 원하는 프로그램은 정수형 집합 두 개를 입력 받고 입력 받은 두 개의 집합을 이용하여 합집합, 교집합, 차집합 연산을 수행하고 그 결과를 출력하는 것이다. 이 때 중요한 것은 집합의 원소는 중복될 수 없다는 것이다.

여러분은 전체 프로그램 코드를 작성하지 않고, 주어진 프로그램을 완성한다. 가정 먼저 알아야할 것은 집합을 위한 변수에 관한 것이다. 집합의 최대 크기는 MAX\_SET\_SIZE로 정의되고, 입력받을 집합 두 개를 각각 setA, setB라 하자. 그리고 합집합, 교집합, 차집합의 결과를 저장하기 위한 변수를 setC로 정의한다. 그리고 sizeA, sizeB, sizeC는 각 집합의 크기이다.

#define MAX_SET_SIZE 10 #define HAVE_ELEMENT 1 #define DO_NOT_HAVE_ELEMENT 0	int setA[MAX_SET_SIZE]; int setB[MAX_SET_SIZE]; int setC[MAX_SET_SIZE*2];  int sizeA; int sizeB; int sizeC;
--	---

그리고 필요한 함수와 각각의 기능에 대한 설명은 아래와 같다.  
다음의 단계로 프로젝트를 수행한다.

#### 단계1:

hasElement는 이미 주어졌고  
printSet, addOneElement는 이전의 문제에서 정의한 함수를 그대로 사용한다.  
printSet, addOneElement를 사용하여 프로그램을 실행시켜보자.

```
int hasElement(int set[], int size, int element);  
- 이 함수는 집합 set에 정수 원소 element가 포함되어 있는지 여부를 검사하는 함수이다. 만약 set에  
  element가 포함되어 있으면HAVE_ELEMENT를 리턴하고, 포함되어 있지 않으면 DO_NOT_HAVE_ELEMENT를 리턴한다.  
  DO_NOT_HAVE_ELEMENT와 HAVE_ELEMENT는 위에 정의되어 있다.  
void printSet(int set[], int size);  
- 이 함수는 주어진 집합을 집합의 형식에 맞게 출력하는 함수이다. 집합의 원소가 1,2,3 이라면 출력은  
  {1, 2, 3}으로 한다.  
int addOneElement(int set[], int size, int element);  
- set에 element가 없으면 이를 추가하고, 이미 있으면 redundant( 혹은 중복입력)라는 메시지를 출력한다.  
  증가된 혹은 유지된 set의 크기를 반환한다.
```

```
C:\Windows\system32\cmd.exe  
Enter the size of Set A:4  
Enter the number for Set A <1/4>:1  
Enter the number for Set A <2/4>:2  
Enter the number for Set A <3/4>:2 -- 중복입력  
It is redundant. Please retry.  
Enter the number for Set A <3/4>:3  
Enter the number for Set A <4/4>:4  
Enter the size of Set B:3  
Enter the number for Set B <1/3>:1  
Enter the number for Set B <2/3>:1 -- 중복입력  
It is redundant. Please retry.  
Enter the number for Set B <2/3>:3  
Enter the number for Set B <3/3>:5  
Set A: < 1, 2, 3, 4 >  
Set B: < 1, 3, 5 >  
Union of setA and setB: < >  
Intersection of setA and setB: < >  
Set-theoretic difference of setA and setB <setA - setB>: < >  
계속하려면 아무 키나 누르십시오 . . .
```

아직 구현 안됨!!

#### 단계2:

이 프로젝트의 과제는 setUnion, setIntersection, setComplement를 작성하는 것이다.

```
int setUnion(int set1[], int size1, int set2[], int size2, int setResult[]);  
- 이 함수는 주어진 집합 set1과 set2의 합집합을 구하는 것이고, 결과는 setResult이라는 배열에 저장된다.  
  그리고 합집합의 크기가 리턴된다. size1과 size2는 각각 set1과 set2 집합의 크기를 의미한다.  
int setIntersection(int set1[], int size1, int set2[], int size2, int setResult[]);  
- 이 함수는 주어진 집합 set1과 set2의 교집합을 구하는 것이고, 결과는 setResult이라는 배열에 저장된다.  
  그리고 교집합의 크기가 리턴된다. size1과 size2는 각각 set1과 set2 집합의 크기를 의미한다.  
int setComplements(int set1[], int size1, int set2[], int size2, int setResult[]);  
- 이 함수는 주어진 집합 set1과 set2의 차집합을 (set1-set2) 구하는 것이고, 결과는 setResult이라는 배열에 저장된다.  
  그리고 차집합의 크기가 리턴된다. size1과 size2는 각각 set1과 set2 집합의 크기를 의미한다. main 함수와 hasElement, printSet 함수는 이미 주어진다. 합/교/차집합 연산을 구현하라.
```

실행예:

```
C:\Windows\system32\cmd.exe  
C:\Users\Staff\Documents\Project1>Project1.exe  
Enter the size of Set A:4  
Enter the number for Set A <1/4>:1  
Enter the number for Set A <2/4>:2  
Enter the number for Set A <3/4>:2 -- 중복 입력  
It is redundant. Please retry.  
Enter the number for Set A <3/4>:3  
Enter the number for Set A <4/4>:4  
Enter the size of Set B:3  
Enter the number for Set B <1/3>:1  
Enter the number for Set B <2/3>:2  
Enter the number for Set B <3/3>:2  
It is redundant. Please retry.  
Enter the number for Set B <3/3>:3  
Set A: < 1, 2, 3, 4 >  
Set B: < 1, 2, 3 >  
Union of SetA and SetB: < 1, 2, 3, 4 >  
Intersection of SetA and SetB: < 1, 2, 3 >  
Set-theoretic difference of SetA and SetB <SetA - SetB>: < 4 >
```

```

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

#define MAX_SET_SIZE 10
#define HAVE_ELEMENT 1
#define DO_NOT_HAVE_ELEMENT 0

int hasElement(int set[], int size, int element);
int setUnion(int set1[], int size1, int set2[], int size2, int setResult[]);
int setIntersection(int set1[], int size1, int set2[], int size2, int setResult[]);
int setComplements(int set1[], int size1, int set2[], int size2, int setResult[]);
void printSet(int set[], int size);
int addOneElement(int set[], int size, int element);

int main(int argc, char *argv[])
{
    int i;

    int setA[MAX_SET_SIZE];
    int setB[MAX_SET_SIZE];
    int setC[MAX_SET_SIZE*2];

    int sizeA;
    int sizeB;
    int sizeC;

    printf("Enter the size of Set A:");
    scanf("%d",&sizeA);
    i = 0;
    while( i < sizeA )
    {
        printf("Enter the number for Set A (%d/%d):", i+1,sizeA );
        scanf( "%d", &setA[i] );
        i = addOneElement(setA, i, setA[i]);
    }

    printf("Enter the size of Set B:");
    scanf("%d",&sizeB);
    i=0;
    while( i < sizeB )
    {
        printf("Enter the number for Set B (%d/%d):", i+1, sizeB );
        scanf( "%d", &setB[i] );
        i = addOneElement(setB, i, setB[i]);
    }

    printf("Set A: ");
    printSet( setA, sizeA );
    printf("Set B: ");
    printSet( setB, sizeB );

    sizeC = setUnion( setA, sizeA, setB, sizeB, setC ); // Union, setC is the result set
    printf("Union of setA and setB: ");
    printSet( setC, sizeC );

    sizeC = setIntersection( setA, sizeA, setB, sizeB, setC ); //Intersection, setC is the result set
    printf("Intersection of setA and setB: ");
    printSet( setC, sizeC );

    sizeC = setComplements( setA, sizeA, setB, sizeB, setC ); //Complements, setC is the result set
    printf("Set-theoretic difference of setA and setB (setA - setB): ");
    printSet( setC, sizeC );

    return 0;

```

```

}

// If the set has the element, returns 1;
// else return 0;
int hasElement(int set[], int size, int element)
{
    int i = 0;
    for( i = 0; i < size; i++ )
        if( set[i] == element )
            return HAVE_ELEMENT; // we found it!

    return DO_NOT_HAVE_ELEMENT;
}
// 원소를 집합에 추가. 이미 존재하면 추가하지 않고 redundant라고 출력한다. 현재의 집합 크기를 반환한다.
int addOneElement(int set[], int size, int element)
{
}

void printSet(int set[], int size)
{
}

}
int setUnion(int set1[], int size1, int set2[], int size2, int setResult[])
{
    // 구현 필요
}
int setIntersection(int set1[], int size1, int set2[], int size2, int setResult[])
{
    // 구현 필요
}
int setComplements(int set1[], int size1, int set2[], int size2, int setResult[])
{
    // 구현 필요
}

```