연습문제

5-1. x^n(x의 n승)을 계산하는 double power(double x, int n) 함수를 작성하여라. 그리고, 3.5^7이 6433.9296875 임을 확인해 보아라.

C code

```
#include <stdio.h>
#include <math.h>
float power(float x,int n);
int main(void)
{
 int n;
 float x;
 printf("x^n ₩n");
 printf("Input x and n => ");
   scanf("%f%d",&x,&n);

 printf("%f₩n",power(x,n));
return 0;
}
float power(float a,int b)
{ int i;
   float po = 1.0;

   for( i = 0; i < b; i++ )
   {   po = po * a; }
 return po;
 }
```

Result (결과)

```
[romance@161s ch5_code]$ ./a.out
x^n
Input x and n => 3.5 7
6433.929688
[romance@161s ch5_code]$
```

5-2. 라이브러리 함수 sqrt() 를 사용하여 정수 인자 k의 네제곱근을 계산하는 함수를 작성하여라. 리턴 되는 값은 double 이어야 한다. 또한 이 함수를 사용하여 여러 값을 표 형식으로 출력하는 프로그램을 작성하여라.

Sol)

```
//간단하게 10과 2의 네제곱근을 구하는 함수를 구했다.
double squareforth(double x)
{ return sqrt(sqrt(x)); }
// 함수를 출력하는 간단한 프로그램.
#include <stdio.h>
#include <math.h>
int main(void)
{
double x,y,z;
printf("%10s  =  %.5f₩n","10^-4",squareforth(10.0));
printf("%10s  =  %.5f₩n","2^-4",squarerforth(2.0));
return 0;
}
```

5-3. 다음 프로그램의 출력을 쓰고, 그 이유를 설명하여라.

```
#include <stdio.h>
int z;

void f(int x)
{ x = 2;
  z + =2;
}

int main(void)
{
z = 5;
f(z);
printf("z = %d₩n",z);
return 0;
}
```

Sol) 답은 7이다. z가 main 밖에서 선언되어서 static 한 역할을 한다.
따라서 f(z)에 의해 바뀐 z값이 main에 그대로 계속 적용이 되어서.
z는 7을 출력하게 된다.

5-4. 전통적인 C의 일반적인 함수 정의 형태는 다음과 같다.

```
Type function_name ( parameter  list)
Declarations of the parameters
{
  declarations
  statsments
 }
```

5.1절 “함수정의”를 참고하여, 연습문제 3번의 f()함수의 함수 정의를 이 형태로 다시 작성하여라.

Sol)

```
Void f( x )    int x;
 {   x =2; z + =2; }
```

5-5. 이번 연습문제는 함수 선언에 관한 것이다. 여기서는 수학 라이브러리에 있는 pow ()함수를 사용 할 것인데, math.h 헤더 파일을 포함시키는 대신 함수 선언을 직접 할 것 이다. 다음 프로그램을 수행해 보아라.

```
#include <stdio.h>
double pow(double,double);
int main(void)
{
 printf("pow(2.0,3.0) = %g₩n",pow(2.0,3.0));
return 0;
}
```

```
[romance@161s ch5_code]$ ./a.out
pow(2.0,3.0) = 8
[romance@161s ch5_code]$
```

그 다음 printf() 문을 다음과 같이 수정한 후 다시 수행해 보아라.

```
 Printf(“pow(2,3) = %g₩n”,pow(2,3));
```

```
[romance@161s ch5_code]$ ./a.out
pow(2,3) = 8
[romance@161s ch5_code]$
```

컴파일러는 double형 인자가 필요하다는 것을 알고 있기 때문에, 제공되는 값을 정확한 형태로 변환시킬 수 있다. 다음으로 함수 원형을 다음과 같이 수정하여 수행해 보아라.

```
 Double pow();
```

```
[romance@161s ch5_code]$ ./a.out
pow(2,3) = 1
[romance@161s ch5_code]$
```

마지막으로 pow() 함수의 선언을 완전히 제거하여라.

```
[romance@161s ch5_code]$ ./a.out
pow(2,3) = -1.99871
[romance@161s ch5_code]$
```

C 시스템이 라이브러리에 있는 함수들의 목적코드를 제공하지만, 사용하고자 하는 함수의 정확한 함수 원형을 제공하는 것은 프로그래머의 책임이다.

5-6. 이번 연습 문제에서는 단정에 대하여 알아본다. 다음 프로그램을 수행하여 그 효과를 이해하여라.

```
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <time.h>
int main(void)
{
int a,b,cnt = 0,i;
 srand(time(NULL));
for (i=0;i<1000;i++) {
 a= rand() %3 + 1;
 b =rand() % 30 + 1;
 if (b-a <=1) continue;
 assert(b-a>2);
 printf("%3d₩n",++cnt);
}
return 0;
}
```

a와 b가 어떤값을 가졌을 때 단정이 실패하는가? 평균적으로, 이 루프는 몇 번 수행되겠는가?

Sol) a와 b의 차가 2이하 일때 단정에 실패한다. 가능한 쌍을 나타내면
(1,1) (1,2) (1,3) (2,1) (2,2) (2,3) (2,4) (3,1) (3,2) (3,3) (3,4) (3,5)
의 경우가 있고 전체 경우의 수는 3*30 = 90가지 이다. 따라서
평균적으로 이 루프는 (1 – 12/90)의 확률을 가지고 돌게 된다.

5-7. 5.12절의 probability() 함수를 사용하여 두개의 파일을 생성하여라, 여기서 한 파일은 100개의 난수를 가지고, 다른 파일은 1000개의 난수를 갖는다. 모든 수는 0과 1사이의 값일 ㄱ서이다. 값이 정말로 무작위 분포를 갖는다면, 난수의 개수가 많아질수록 그 값들의 평균은 0.5에 근접하게 될 것이다. 각 파일에 있는 난수들의 평균을 계산하여라. 일반적으로 난수 1000개의 평균의 난수 100개의 평균보다 0.5에 가가울 것이다. 과연 그러한가?

Sol) 예

5-8. x에 대한 2차 다항식은 다음과 같다.
ax^2+bx+c
임의의 2차 다항식의 값을 계산하는 다음과 같은 형태의 함수를 작성하여라.
Double f(double a, double b, double c, double x)

Sol) double f(double a, double b,double c , double x)
{ return a*a+ b*x+ c; }

5-9. 5.4절의 tbl_of_powers 프로그램을 실행시켜 보아라. 부정확한 값이 출력되기 까지 몇 개의 행이 계산되는가? Double 형으로 수정하여 다시 수행해 보아라. 그 행의 개수가 더 커지는가?

Sol) 예.

5-10. 5.4절의 tbl_of_power 프로그램에서 main() 함수 정의를 먼저 작성했다. 함수 원형을 다 지우고 main() 함수를 맨 뒤 작성하면 어떤 일이 발생하는가? 컴파일러는 별 문제 없이 수행 될 것이다. 그러한가? 함수 원형을 다 지운 상태에서 main() 함수를 맨 앞으로 옮기면 어떻게 되겠는가? 어떤 C 컴파일러는 경고 메시지를 출력하지만, 실행 파일은 만든다. 반면에 C++ 컴파일러는 오류 메시지를 출력하고, 더 이상 컴파일 되지 않는다.

Sol)

Gcc 에서 컴파일 했을 때 ::

```
[romance@161s ch5_code]$ cc 5_10.c
5_10.c:15: warning: type mismatch with previous implicit declaration
5_10.c:10: warning: previous implicit declaration of `prn_heading'
5_10.c:15: warning: `prn_heading' was previously implicitly declared to return `int'
5_10.c:18: warning: type mismatch with previous implicit declaration
5_10.c:11: warning: previous implicit declaration of `prn_tbl_of_powers'
5_10.c:18: warning: `prn_tbl_of_powers' was previously implicitly declared to
```

```
return `int'
5_10.c:27: warning: type mismatch with previous implicit declaration
5_10.c:21: warning: previous implicit declaration of `power'
5_10.c:27: warning: `power' was previously implicitly declared to return `int'
[romance@161s ch5_code]$

C++ 에서 컴파일 했을 때 ::

[romance@161s ch5_code]$ c++ 5_10.c
5_10.c: In function `int main ()':
5_10.c:10: `prn_heading' undeclared (first use this function)
5_10.c:10: (Each undeclared identifier is reported only once for each
function it appears in.)
5_10.c:11: `prn_tbl_of_powers' undeclared (first use this function)
5_10.c: In function `void prn_heading ()':
5_10.c:15: `void prn_heading ()' used prior to declaration
5_10.c: In function `void prn_tbl_of_powers (int)':
5_10.c:18: `void prn_tbl_of_powers (int)' used prior to declaration
5_10.c:21: `power' undeclared (first use this function)
5_10.c: In function `long int power (int, int)':
5_10.c:27: `long int power (int, int)' used prior to declaration
[romance@161s ch5_code]$
```

5-11. n이 소수이면 1을 리턴하고, 아니면 0을 리턴하는 int is_prime(n) 함수를 작성하여라.

Sol)

C code

```
#include <stdio.h>
#include <limits.h>
int is_prime(int);
int main(void)
{
int x;
```

```
scanf("%d",&x);
if (is_prime(x)) { printf(" PRIME!! ₩n"); }
else { printf("NOT PRIME !! ₩n"); }
return   0;
}

int is_prime(int n)
{ int i;
   for (i = 2; i <n; + + i)
   { if ( n % i == 0 && n != i) return 0;
   } return 1;
}
```

Result (결과)

```
[romance@161s ch5_code]$ ./a.out
2
 PRIME!!
[romance@161s ch5_code]$ ./a.out
3
 PRIME!!
[romance@161s ch5_code]$ ./a.out
4
NOT PRIME !!
[romance@161s ch5_code]$
```

5-12. n번째 Fibonacci 수가 소수인가를 검사하는 is_fib_prime(n) 함수를 작성하여라. 이 함수는 두 함수를 호출해야 한다. 하나는 4.13절에서 소개한 반복적 함수인 fibonacci()이고, 다른 하나는 앞의 연습문제에서 작성한 is_prime() 함수이다. 3과 10 사이의 n에 대해서, n번째 fibonacci 수가 소수일 필요 충분 조건은 n이 소수인 것이다. N이 10보다 클 때 is_fib_prime()함수는 어떻게 동작하는지 조사하여라.

Sol)

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

```
int is_prime(int);
int fibonacci(int);
int main(void)
{
int x;
scanf("%d",&x);
if (is_prime(fibonacci(x))) { printf("FIBONACCI PRIME!! ₩n"); }
else { printf("FIBONACCI NOT PRIME !! ₩n"); }
return  0;
}

int is_prime(int n)
{ int i;
  for (i = 2; i <n; + + i)
  { if ( n % i == 0 && n != i) return 0;
  } return 1;
}
int fibonacci(int n)
{
  if (n<=1) return n;
  else return (fibonacci(n-1)+ fibonacci(n-2));
}
```

5-13. Goldbach 추측이라고 하는 유명한 추측은 “2보다 큰 모든 짝수는 두개의 소수의 합으로 표현될 수 있다”라는 것이다. 이 추측을 검사하기 위해 컴퓨터를 많이 사용하고 있다. 반례는 아직 발견되고 있지 않았다. 기호 상수 START와 FINISH 사이의 모든 짝수에 대하여 이 추측이 맞는지 검사하는 프로그램을 작성하여라.

Sol)

C code

```
#include <stdio.h>
#include <limits.h>
int is_prime(int);
```

```
int main(void)
{
int START;
int FINISH;
int i,j,k=0;
scanf("%d",&START);
scanf("%d",&FINISH);
for (i = START; i <= FINISH; i+ =2)
 {
    for ( j=2;j<i; + + j)
        {   if (is_prime(j) && is_prime(i-j)) { k = 1;
                                                printf   ("     %d   =   %d   +   %d
₩n",i,j,i-j);
                                                break; }
        } if (k == 0) printf(" NOT PROVE ₩n" );
    k = 0;
 }
}
int is_prime(int n)
{ int i;
  for (i = 2; i <n; + + i)
  { if ( n % i == 0 && n != i) return 0;
  } return 1;
}
```

Result (결과)

```
[romance@161s ch5_code]$ ./a.out
700
1000
  700 = 17 +  683
  702 = 11 +  691
  704 = 3 +  701
  706 = 5 +  701
  708 = 7 +  701


  .....
```

```
988 = 5 +  983
   990 = 7 +  983
   992 = 73 +  919
   994 = 3 +  991
   996 = 5 +  991
   998 = 7 +  991
   1000 = 3 +  997
[romance@161s ch5_code]$
```

5-14. 주어진 숫자에 대한 모든 인수를 찾는 함수를 작성하여라. 예를 들면, 다음과 같다.
9 = 3 * 3, 17 = 17(소수), 52 = 2 * 2 * 13
작은 숫자의 인수를 찾는 것은 쉽고, 인수를 찾는 프로그램을 작성하는 것도 문제가 없을 것이다. 그러나 일반적으로 인수를 구하는 것은 매우 어렵다. 몇백 자리로 이루어진 정수의 인수를 찾는 것은 대형 컴퓨터를 사용해도 불가능 하다.

Sol)

C code

```
#include <stdio.h>
#include <limits.h>
void factor(int);
int main(void)
{
int a,i;
scanf("%d",&a);
factor(a);

 }




void factor(int n)
{
```

```
  int i=2;
  int k = n;
 printf(" %d = ",k);
 while (k != 1)
  {
      if ( k % i == 0 ) { k = k / i;
                          printf(" %d * ",i);
                        }
       else if (i != n) {++i;}

  }
  printf(" 1₩n");
}
```

Result (결과)

```
[romance@161s ch5_code]$ ./a.out
10
 10 =  2 *  5 *  1
[romance@161s ch5_code]$ ./a.out
5
 5 =  5 *  1
[romance@161s ch5_code]$ ./a.out
20
 20 =  2 *  2 *  5 *  1
[romance@161s ch5_code]$ ./a.out
17
 17 =  17 *  1
[romance@161s ch5_code]$
```

5-15. 이 연습문제는 식별자의 유효 범위에 관한 것이다. 다음 코드는 어떤 값을 출력하겠는가? 먼저 답을 작성하고, 프로그램을 작성한 후에 답을 검사해 보아라.

```
int a=1,b=2,c=3;
a + =b+ = + + c;
printf("%5d%5d%5d₩n",a,b,c);
{
  float b = 4.0;
  int c;
  a + = c = 5*b;
  printf("%5d%5.1f%5d₩n",a,b,c);
}
printf("%5d%5d%5d₩n",a,b,c);
```

Sol)

{ } 안에서 선언된 변수는 거기에서만 유효 하므로 다음과 같은 출력을 보일것이다.

7 6 4

27 4.0 20

27 6 4

Result (결과)

```
[romance@161s ch5_code]$ ./a.out
    7     6     4
   27   4.0    20
   27     6     4
[romance@161s ch5_code]$
```

5-16. 5.14절의 “The universe is never ending !” 프로그램을 수정하여 17번 호출한 후 종료하도록 하여라. 수정한 프로그램은 하나의 main()함수만으로 구성되어야 하며, 재귀 호출해야 한다.

Sol)

C code

```
#include <stdio.h>
int x = 1;

int main(void)
{
printf(" The universe is nert ending ! ");
if ( x == 17 ) {   return 0; }
else { + + x; main(); }
return   0;
}
```

5-17. 다음 함수는 어떤 시스템에서 부정확한 값을 생성할 것이다.

```
int factorial(int n)
{
  if (n ==0 || n ==1)
  return 0;
  else return (n * factorial (--n));
}
```

이 프로그램을 수행해 보고, 함수가 생성하는 값이 왜 시스템 의존적인지 설명해 보아라.

Sol)

마지막에서 두번째 라인에서 factorial(--n) 구문에서 --n을 먼저 고려를 하면 이미 n은 n-1된 상태로 된다. 즉 return ((n-1)*factorial(n-1)); 이 되는 셈이다. 따라서 --n을 어떻게 처리하냐에 따라서 시스템 의존 적이다.

5-18. 두 양의 정수의 최대공약수는 두 수의 공통 약수 중 가장 큰 수이다. 예를 들어 , 3은 6과 15의 최대 공약수이고, 1은 15와 22의 최대공약수이다. 다음은 두 양의 정수의 최대 공약수를 계산하는 재귀적 함수이다.

Sol)

```
#include <stdio.h>
int re_gcd(int,int);
int seq_gcd(int,int);
int main(void)
{
int x,y,z,w;
scanf("%d",&x);
scanf("%d",&y);
z =re_gcd(x,y);
w =seq_gcd(x,y);
if (z == w) { printf("OK!₩n");}
printf("recursive function ::   %d ₩n ",z);
printf("sequantional function :: %d ₩n",w);
return   0;
}
int re_gcd(int p, int q)
{ int r;
   if ((r = p % q) == 0) return q;
   else return re_gcd(q,r);
}

int seq_gcd(int p, int q)
{ int r;
   while ( r != 0 )
   {
      r = p % q;
      p = q;
      q = r;
   }
   return q;
}
```

5-19. 어떤 시스템에서 키워드 extern은 표준 헤더 파일의 함수 선언과 함수 원형에 사용된다. 이것은 전통적인 C시스템에서 일반적이지만, ANSI C 시스템에서는 그렇지 않다. 여러분의 시스템은 이렇게 되어 있는가?

Sol) 함수 선언과 함수 원형에 사용된다.

5-20. 다음 프로그램은 고의적으로 함수의 하단에 외부 변수를 선언한 것이다. 어떤 값이 출력되는가?

```
#include <stdio.h>

int main(void)
{
 extern int a,b,c;
 printf("%3d%3d%3d₩n",a,b,c);
 return 0;
}
int a = 1,b=2,c=3;
```

```
[romance@161s ch5_code]$ ./a.out
  1  2  3
[romance@161s ch5_code]$
```

이 프로그램의 마지막 줄을 다음과 같이 수정하여라.

```
static int a= 1 , b=2, c=3;
```

이제 파일의 하단에 있는 변수들은 정적 외부 변수이므로, 이 변수들은 main()에서 사용될 수 없다. 그래서, main()에서 참조하는 외부 변수를 찾을 수 없고, 컴파일러는 오류 메시지를 출력할 것이다. 그러한가?

```
[romance@161s ch5_code]$ cc 5_20_a.c
5_20_a.c:9: warning: `a' was declared `extern' and later `static'
5_20_a.c:9: warning: `b' was declared `extern' and later `static'
5_20_a.c:9: warning: `c' was declared `extern' and later `static'
[romance@161s ch5_code]$
```

5-21. 전통적인 C에서 변수를 선언할 때, 기억 영역 클래스 명시자와 형 명시자를 임의의 순서대로 기술하는 것이 가능하다. 예를 들어, 다음 두 선언은 같은 의미이다.

register int i;
int register I;

ANSI C는 기억영역 클래스 명시자를 먼저 쓰도록 되어 있다. 그럼에도 불구하고, 대부분의 ANSI C 컴파일러는 두가지 다 처리 할것이다. 여러분의 컴파일러는 역순으로 기술 된 것을 처리할 수 있는지 검사해 보아라.

Sol)

C code

```
#include <stdio.h>

int main(void)
{
int register i;
register int x;

return  0;
}
```

Result (결과)

```
[romance@161s ch5_code]$ cc 5_21.c
[romance@161s ch5_code]$ ./a.out
[romance@161s ch5_code]$
```

5-22. 다음 프로그램이 어떻게 동작하는지 설명해 보아라.

```
#include <stdio.h>
#include <stdlib.h>
#define FOREVER 1
#define STOP 17

int main(void)
{
void f(void);
   while (FOREVER)
    f();
   return 0;
}
void f(void)
{
 static int cnt = 0;
printf("cnt = %d₩n",++cnt);
if (cnt == STOP) exit(0);
}
```

Sol)

Main 함수 안에서 f가 while 무한 루프로 들어가지만 cnt가 1씩 증가하면서 17이 되면 exit(0) 함수로 프로그램을 빠져 나간다.

```
[romance@161s ch5_code]$ ./a.out
cnt = 1
cnt = 2
cnt = 3
cnt = 4
cnt = 5
cnt = 6
cnt = 7
cnt = 8
cnt = 9
cnt = 10
cnt = 11
```

```
cnt = 12
cnt = 13
cnt = 14
cnt = 15
cnt = 16
cnt = 17
[romance@161s ch5_code]$
```

5-23. n_0을 주어진 양의 정수라 하자. 이때 n_{i+1}은, I = 0,1,2, …에 대해서, 다음과 같이 정의 된다.

n_i가 짝수이면, $n_{i+1}=n_i/2$;

홀수이면 3n+ 1

이 수열은 n_i 값이 1일때 멈추게 된다. 이런 방법으로 생성되는 수를 “hailstones” 라고 한다. Hailstones를 생성하는 프로그램을 작성하여라. 함수 헤드는 다음과 같다

```
void hailstones(int n)
{            ....   }
```

Sol)

```
#include <stdio.h>
void hailstones(int);
int n;
int main(void)
{
int x;
scanf("%d",&x);
hailstones(x);
return   0;
}
void hailstones(int n)
{
int num=1;
  while (n != 1)
{
  if (n % 2 == 0 ) {n = n/2; }
  else { n = 3*n + 1; }
```

```
  if (num % 6 == 0) printf("%7d₩n",n);
  else printf("%7d",n);
  ++num;
}
      printf("₩n Number of hailstones generated₩n",num);
}
```

5-24. 표준 라이브러리에 있는 난수발생기 rand()를 사용하여 동전던지기를 모의 실험하는 프로그램을 작성하여라. 수식 rand() % 2를 사용하여 int 값 0이나 1을 생성하여라.

C code

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(void)
{
int a,i;
srand(time(NULL));

for (i=0; i<1000; ++i)
{
  a = rand()%2;
  printf(" %d",a);
  if (i % 30 == 0 ) printf("₩n");
}
printf("₩n");
return  0;
}
```

Result (결과)

```
[romance@161s ch5_code]$ ./a.out
 0
```

```
0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 0 1 0 0 1 1 1 1 1 0
0 1 0 0 0 1 1 0 1 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 1 0 1 0 0 0
1 0 0 1 0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 0 0 1 0 0 1 1 0 1 0 0
1 1 0 1 0 1 1 1 0 0 0 1 0 1 1 0 1 0 1 1 0 1 0 0 1 0 0 0 1 0
0 0 1 1 0 1 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 1 1 0 0 1 0 0 0
0 1 1 1 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 1 1 0 1 0 0 0
1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0
0 1 0 1 1 1 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0
0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 1 0 1 1 0 0 0 0
0 0 0 0 1 0 0 0 1 1 0 1 1 0 0 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1
0 1 1 0 1 0 1 1 1 0 0 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 0 0 0 0
1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 1 0 1 1 1 0 1 0 1 1 1 1 1 1 0
1 0 0 1 1 1 1 0 1 1 0 1 0 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1
0 0 1 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0
0 0 0 0 1 0 0 1 1 1 1 1 1 1 1 0 1 1 0 0 1 0 0 1 0 1 1 1 0 1 1
0 1 1 0 1 0 0 0 1 1 1 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 0 1 0
1 1 0 0 1 1 0 0 1 1 1 0 0 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 1
0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 0 0
0 0 0 0 0 1 0 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 0 1 1
0 1 1 1 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 1 1 0
0 1 0 1 0 0 1 0 1 1 1 1 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 1
0 0 1 0 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1
0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 0 0 0 0 1 1 0 1 1 0 1 0
1 1 1 1 0 0 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0
0 1 1 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 1 1 1 1 1 0 1 0 0 1 0 1
1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 1 0 0 1 1 0
0 1 0 1 0 1 1 1 1 0 1 0 0 1 1 0 1 0 0 0 1 1 1 0 0 0 0 0 1 0
1 1 1 1 0 1 1 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 1 0 0 1 0 0 0 1
0 1 1 1 0 1 0 1 1 0 0 1 1 0 1 1 0 1 0 0 1 0 0 1 0 0 0 1 0 0
0 1 1 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 1 1 0 0 1
0 1 0 0 0 1 0 1 0 1 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 0 0 0 0 1
0 1 0 0 1 0 1 1 0 1 0 0 1 1 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 0
1 0 1 1 1 0 1 0 1
[romance@161s ch5_code]$
```

5-25. rand() 함수가 아닌 다른 난수 발생기를 사용할 수 있다면, 앞의 연습문제에서 작성한 프로그램이 그 함수을 사용할 경우 어떻게 되는지 실험해 보아라. 실험결과는 다른가?

Sol) lrand48() 같은 함수를 사용가능 하다.

5-26. 확률적 사건에 대하여 난수발생기를 연속적으로 사용하는 모의 실험을 몬테카를로 시뮬레이션이라고 한다.
Sol) 생략

5-27. 하노이 탑 프로그램을 수행해 보아라. N개의 원반이 A탑에 있을 때, 이 문제를 풀기 위해 몇번의 이동이 필요한가? N이 64일때, 매일 한 원반만 이동한다면, 모든 원반을 이동하기 위해 몇 년이 걸리겠는가? 만일 수도사가 1초에 원반 하나를 옮길 수 있다면, 세상이 끝나기 전에 모든 원반을 옮길 수 있을까?
Sol)

```
#include "5_27.h"

int cnt = 0;
int main (void)
{
  int n;
  scanf("%d",&n);
  move(n,'A','B','C');
  printf("%d",cnt);
  return 0;
}

int get_n_from_user(void)
 { int n;
      if(scanf("%d",&n) != 1 || n < 1) {
                printf("₩nERROR : Positive integer not found - bye!₩n₩n");
                exit(1);
     }printf("₩n");
     return n;
```

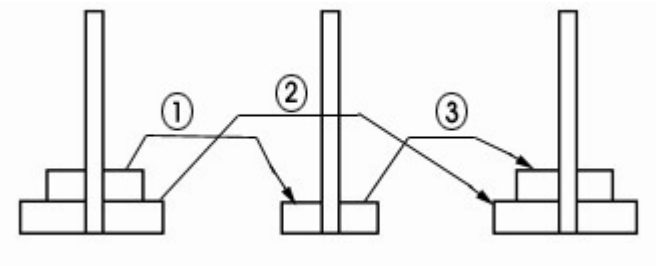
```
}


void move(int n, char a,char b,char c)
{
 if(n==1) {
  ++cnt;
  } else {move(n-1,a,c,b);
            ++cnt;
            move (n-1,b,a,c);
  }
}
```

Result (결과)

```
[romance@161s ch5_code]$ cc 5_27.c
[romance@161s ch5_code]$ ./a.out
64
 .... (너무 오래 걸림)
```

Additonal Text

하노이 탑의 수학적 원리 :



위의 경우는 원판의 개수가 2 이다. 이 그림에서 보면은 최소한 3 번 이동하면, 다른 원판으로 옮겨짐을 알 수가 있다.

그리고, 원판의 개수가 3 이면, 최소이동 수는 7 번이 됨을 알 수가 있다.

다시 정리하면, 원판의 개수를 n 이라 하고, 최소이동수를 M 이라 하면,
n = 2, M = $2^2 - 1$; n = 3, M = $2^3 - 1$이러한 식으로부터 추측을 하면,
64 개의 원판을 옮기는 데는 $2^{64} - 1$번이라는 것을 알 수가 있다.

전설에서처럼 64 개를 옮길 때, 한 번 옮기는 시간을 1 초 걸린다고 하면 64 개를 옮기는 데는 $2^{64} - 1$초가 걸린다는 것을 알 수 있고, 이를 년으로 환산하면, 대략 5833 억년 정도 하고 한다.

Chapter 5 End point.