연습문제

3-1. 모든 실수를 컴퓨터에서 표현할 수 있는 것은 아니다. 따라서 컴퓨터에서 이용 가능한 실수는 매우 일부분이다. 이러한 예로서 다음 코드를 수행시키면, 두개의 일한 수를 출력할 것이다.

```
double x = 123.45123451234512345;
double y = 123.45123451234512300;
printf ("%.17f %17f₩n",x,y);
```

Sol)

```
[romance@161s ch3_code]$ ./a.out
123.45123451234512402 123.45123451234512402
[romance@161s ch3_code]$
```

3-2. 다음 수학 공식은 모든 x에 대하서 만족한다.

$$\sin^2(x) + \cos^2(x) = 1$$

다음 프로그램을 수행하여 컴퓨터에서 이것이 항상 만족하는지 검사해 보아라.

```
#include <math.h>
#include <stdio.h>
int main(void)
{
double two_pi = 2.0 * M_PI;
double h = 0.1;
double x;
 for (x =0.0; x<two_pi; x + =h)
  printf ("%5.1f: %.15e₩n",x,sin(x)*sin(x)+ cos(x)*cos(x));
  return 0;
}
```

Sol) 생략

3-3. sin(), cos(),tan() 에 대한 삼각함수 값을 표로 출력하는 프로그램을 작성하여라.
표에서 각도는 0에서 π까지 20단계로 한다.

C code

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
double h = M_PI/20;
double x;
 for (x =0.0; x< M_PI; x + = h)
printf ("x = %.15e > sin(x) = %.15e cos(x) = %.15e tan(x) = %.15e",x,
sin(x),cos(x),tan(x));

return 0;
}
```

Result (결과)

```
[romance@161s ch3_code]$ cc 3_3.c -lm
[romance@161s ch3_code]$ ./a.out
x = 0.00000e+00 > sin(x) =  0.00000e+00 cos(x) = 1.00000e+00 tan(x) = 0.00000e+00
x = 1.57080e-01 > sin(x) =  1.56434e-01 cos(x) = 9.87688e-01 tan(x) = 1.58384e-01
x = 3.14159e-01 > sin(x) =  3.09017e-01 cos(x) = 9.51057e-01 tan(x) = 3.24920e-01
x = 4.71239e-01 > sin(x) =  4.53990e-01 cos(x) = 8.91007e-01 tan(x) = 5.09525e-01
x = 6.28319e-01 > sin(x) =  5.87785e-01 cos(x) = 8.09017e-01 tan(x) = 7.26543e-01
x = 7.85398e-01 > sin(x) =  7.07107e-01 cos(x) = 7.07107e-01 tan(x) = 1.00000e-00
x = 9.42478e-01 > sin(x) =  8.09017e-01 cos(x) = 5.87785e-01 tan(x) = 1.37638e+00
x = 1.09956e+00 > sin(x) =  8.91007e-01 cos(x) = 4.53990e-01 tan(x) = 1.96261e+00
x = 1.25664e+00 > sin(x) =  9.51057e-01 cos(x) = 3.09017e-01 tan(x) = 3.07768e+00
x = 1.41372e+00 > sin(x) =  9.87688e-01 cos(x) = 1.56434e-01 tan(x) = 6.31375e+00
x = 1.57080e+00 > sin(x) =  1.00000e+00 cos(x) = 6.12303e-17 tan(x) = 1.63318e+16
x = 1.72788e+00 > sin(x) =  9.87688e-01 cos(x) = -1.56434e-01 tan(x) = -6.31375e+00
```

```
x = 1.88496e+00 > sin(x) =   9.51057e-01 cos(x) = -3.09017e-01 tan(x) = -3.07768e+00
x = 2.04204e+00 > sin(x) =   8.91007e-01 cos(x) = -4.53990e-01 tan(x) = -1.96261e+00
x = 2.19911e+00 > sin(x) =   8.09017e-01 cos(x) = -5.87785e-01 tan(x) = -1.37638e+00
x = 2.35619e+00 > sin(x) =   7.07107e-01 cos(x) = -7.07107e-01 tan(x) = -1.00000e+00
x = 2.51327e+00 > sin(x) =   5.87785e-01 cos(x) = -8.09017e-01 tan(x) = -7.26543e-01
x = 2.67035e+00 > sin(x) =   4.53990e-01 cos(x) = -8.91007e-01 tan(x) = -5.09525e-01
x = 2.82743e+00 > sin(x) =   3.09017e-01 cos(x) = -9.51057e-01 tan(x) = -3.24920e-01
x = 2.98451e+00 > sin(x) =   1.56434e-01 cos(x) = -9.87688e-01 tan(x) = -1.58384e-01
x = 3.14159e+00 > sin(x) =   1.22461e-16 cos(x) = -1.00000e+00 tan(x) = -1.22461e-16
[romance@161s ch3_code]$
```

Additonal Text

Q. C 의 소스를 컴파일 하면(자),

... undefined reference to `sin'

... undefined reference to `XOpenDisplay'

등이라고 하기 에러가 됩니다.

A. 컴파일은 성공해, 오브젝트 파일 (*.o)가 생성되었습니다만,
링크시에 sin·XOpenDisplay 라고 하기 함수가 발견되지 없었다고 하자
에러입니다.

/usr/lib/ 나 /usr/X11R6/lib, /usr/local/lib/ 에는 lib*.so.* (이)나
lib*.a 라고 하기 파일이 있습니다.이것들은 라이브러리라고 말해,
컴파일제의 함수군이 들어가 있습니다.

sin 는 /usr/lib/libm.so.* 에, XOpenDisplay 는 /usr/X11R6/lib/libX11.so.*
에 포함되어 있기 때문에, 이 장소를 가르쳐 주면 좋습니다.
예를 들면, sin 가 발견되지 없으면

% cc foo.c -lm

(으)로 합니다.XOpenDisplay 가 발견되지 없으면

% cc foo.c -lX11 -L /usr/X11R6/lib

(으)로 합니다.

-lm 라고 하는 것은, 링커 (/usr/bin/ld)에
「libm.so.* 그렇다고 하기 라이브러리를 찾으세요」

그렇다고 하기 옵션입니다.

3-4. 소수점 이하 부분을 갖는 float 형이나 double 형을 출력할 때, printf() 함수가 절단을 수행하는지 반올림을 수행하는지를 알아보는 프로그램을 작성하여라. ANSI 표준은 반올림하도록 요구하나, 이것은 시스템 종속적이다. 사용자 컴퓨터에서는 어떻게 수행하는가?

C code

```
#include <stdio.h>
int main (void)
{
double x = 0.55555555555555555;
float   y = 0.55555555555555555;
printf ("%e %f₩n",x,y);
return 0;
}
```

Result (결과)

```
[romance@161s ch3_code]$ ./a.out
5.555556e-01 0.555556                    (반올림을 수행한다.)
[romance@161s ch3_code]$
```

3-5. 2의 거듭제곱 목록을 10진수,16진수, 8진수로 출력하는 프로그램을 다음 코드를 사용하여 완성하여라.

```
int i,val = 1;
for (i = 0; i < 35; + + i) {
    printf("%15d%15u%15x%15o₩n",val,val,val,val,val);
    val *=2; }
```

Sol)

Result (결과)

```
[romance@161s ch3_code]$ ./a.out
```

1	1	1	1
2	2	2	2
4	4	4	4
8	8	8	10
16	16	10	20
32	32	20	40
64	64	40	100
128	128	80	200
256	256	100	400
512	512	200	1000
1024	1024	400	2000
2048	2048	800	4000
4096	4096	1000	10000
8192	8192	2000	20000
16384	16384	4000	40000
32768	32768	8000	100000
65536	65536	10000	200000
131072	131072	20000	400000
262144	262144	40000	1000000
524288	524288	80000	2000000
1048576	1048576	100000	4000000
2097152	2097152	200000	10000000
4194304	4194304	400000	20000000
8388608	8388608	800000	40000000
16777216	16777216	1000000	100000000
33554432	33554432	2000000	200000000
67108864	67108864	4000000	400000000
134217728	134217728	8000000	1000000000
268435456	268435456	10000000	2000000000
536870912	536870912	20000000	4000000000
1073741824	1073741824	40000000	10000000000

```
    -2147483648       2147483648        80000000      20000000000
              0                0               0                0
              0                0               0                0
              0                0               0                0
[romance@161s ch3_code]$
```

2의 제곱승이기 때문에 각각 8진법이나 16진법이나 0을 다수 포함하게 된다.

3-6. 현재 사용하는 시스템에서 다음 코드를 수행하여라.

```
int big_big = 2000000000 +  2000000000;
 printf("%d %u₩n",big_big,big_big);
```

만일 2바이트 워드를 갖는 컴퓨터를 사용한다면, 2000000000을 32000으로 변경하여 여라. 무엇이 출력되는가? 변수 big_big 의 형을 int 형이 아닌 unsigned로 선언하면 출력은 어떻게 변하겠는가?

Sol)

a) 코드 수행시

```
[romance@161s ch3_code]$ cc 3_6.c
3_6.c: In function `main':
3_6.c:4: warning: integer overflow in expression
```

b) int 형을 unsigned로 선언했을 경우

```
3_6_a.c: In function `main':
3_6_a.c:4: warning: integer overflow in expression
[romance@161s ch3_code]$
```

3-7. 다음 코드를 주의 깊게 살혀보아라.

printf (“Why is 21 + 31 equal to %d?₩n”,2l+ 3l);

4바이트 int 컴퓨터에서는 다음과 같은 결과를 얻었다.

Why is 21 + 31 equal to 5 ?

어떻게 이런 출력이 나왔는지를 설명하여라.

Sol)

21 + 31 이 아닌 2l(알파벳 엘) + 3l(알파벳 엘) 이기 때문에

문자를 무시하고 2 + 3 으로 계산이 된다.

3-8. 수학에서 ε(입실론) 은 작은 양수를 나타내는 데 사용된다. 수학에서는 임의의 작은 이라는 ㄱ서이 가능하지만, 컴퓨터에서는 “임의의 작은” 이라는 개념이 없다. 수치 해석에서는 double 형의 변수 esp(spsilon)을 선언을 하고, 다음식을 만족하는 가장 작은 양수를 esp에 할당하는 것이 편리할 때가 있다.

1.0 < 1.0 + esp

이 수치는 컴퓨터에 종속적이다. 사용자의 컴퓨터에서 이 eps 값을 보아라.

eps 값을 1e-37에서부터 시작하여 보아라. 이 값에 대해서 위의 식은 거짓 이다.

C code

```
#include <stdio.h>
int main (void)
{ double esp,x; esp = 1e-37;
while ( 1.0 >= 1.0 + x ) { x = x*10;}
printf ("%e₩n",x);
return 0;    }
```

Result (결과)

```
[romance@161s ch3_code]$ ./a.out
4.854448e-19
[romance@161s ch3_code]$
```

2-9. 두 함수 tan(sin(x)) 와 sin(tan(x)) 를 원점에서 Taylor 전개하면 처음 7개항까지는 같다. 이것은 원점에서 두 함수의 차가 0에 가까움을 뜻한다. 다음 프로그램을 실행시켜 보아라.

```
[romance@161s ch3_code]$ ./a.out
f(-0.25) = -0.000002187066151
f(-0.24) = -0.000001634129080
f(-0.23) = -0.000001206510996
f(-0.22) = -0.000000879278937
f(-0.21) = -0.000000631737860
f(-0.20) = -0.000000446835827
f(-0.19) = -0.000000310635740
f(-0.18) = -0.000000211847706
f(-0.17) = -0.000000141416575
f(-0.16) = -0.000000092159651
f(-0.15) = -0.000000058449944
f(-0.14) = -0.000000035940738
f(-0.13) = -0.000000021327533
f(-0.12) = -0.000000012143758
f(-0.11) = -0.000000006586900
f(-0.10) = -0.000000003371947
f(-0.09) = -0.000000001609264
f(-0.08) = -0.000000000704221
f(-0.07) = -0.000000000276067
f(-0.06) = -0.000000000093699
f(-0.05) = -0.000000000026117
f(-0.04) = -0.000000000005471
f(-0.03) = -0.000000000000730
f(-0.02) = -0.000000000000043
f(-0.01) = -0.000000000000000
f(+0.00) = +0.000000000000000
f(+0.01) = +0.000000000000000
f(+0.02) = +0.000000000000043
f(+0.03) = +0.000000000000730
f(+0.04) = +0.000000000005471
```

```
f(+0.05) = +0.000000000026117
f(+0.06) = +0.000000000093699
f(+0.07) = +0.000000000276067
f(+0.08) = +0.000000000704221
f(+0.09) = +0.000000001609264
f(+0.10) = +0.000000003371947
f(+0.11) = +0.000000006586900
f(+0.12) = +0.000000012143758
f(+0.13) = +0.000000021327533
f(+0.14) = +0.000000035940738
f(+0.15) = +0.000000058449944
f(+0.16) = +0.000000092159651
f(+0.17) = +0.000000141416575
f(+0.18) = +0.000000211847706
f(+0.19) = +0.000000310635740
f(+0.20) = +0.000000446835827
f(+0.21) = +0.000000631737860
f(+0.22) = +0.000000879278937
f(+0.23) = +0.000001206510996
f(+0.24) = +0.000001634129080
[romance@161s ch3_code]$
```

프로그램의 출력을 보고 원점 근처에서 0에 가까움을 확인해 보아라. 출력 형식에서 + 기호를 제거한 후 수행해 보아라.

```
[romance@161s ch3_code]$ ./a.out
f(-0.25) = -0.000002187066151
f(-0.24) = -0.000001634129080
f(-0.23) = -0.000001206510996
f(-0.22) = -0.000000879278937
f(-0.21) = -0.000000631737860
f(-0.20) = -0.000000446835827
f(-0.19) = -0.000000310635740
f(-0.18) = -0.000000211847706
f(-0.17) = -0.000000141416575
f(-0.16) = -0.000000092159651
```

f(-0.15) = -0.000000058449944
f(-0.14) = -0.000000035940738
f(-0.13) = -0.000000021327533
f(-0.12) = -0.000000012143758
f(-0.11) = -0.000000006586900
f(-0.10) = -0.000000003371947
f(-0.09) = -0.000000001609264
f(-0.08) = -0.000000000704221
f(-0.07) = -0.000000000276067
f(-0.06) = -0.000000000093699
f(-0.05) = -0.000000000026117
f(-0.04) = -0.000000000005471
f(-0.03) = -0.000000000000730
f(-0.02) = -0.000000000000043
f(-0.01) = -0.000000000000000
f(0.00) = 0.000000000000000
f(0.01) = 0.000000000000000
f(0.02) = 0.000000000000043
f(0.03) = 0.000000000000730
f(0.04) = 0.000000000005471
f(0.05) = 0.000000000026117
f(0.06) = 0.000000000093699
f(0.07) = 0.000000000276067
f(0.08) = 0.000000000704221
f(0.09) = 0.000000001609264
f(0.10) = 0.000000003371947
f(0.11) = 0.000000006586900
f(0.12) = 0.000000012143758
f(0.13) = 0.000000021327533
f(0.14) = 0.000000035940738
f(0.15) = 0.000000058449944
f(0.16) = 0.000000092159651
f(0.17) = 0.000000141416575
f(0.18) = 0.000000211847706
f(0.19) = 0.000000310635740
f(0.20) = 0.000000446835827

```
f(0.21) = 0.000000631737860
f(0.22) = 0.000000879278937
f(0.23) = 0.000001206510996
f(0.24) = 0.000001634129080
[romance@161s ch3_code]$

( 똑 같은 데... )
```

2-10. 대부분의 컴퓨터는 정수를 저장하는 데 2의 보수 표현을 사용한다. 이러한 컴퓨터에서 –1을 정수적형으로 저장하면 모든 비트는 1이 된다. 사용자 시스템이 이러한 컴퓨터라면, char 형이 signed char 형과 unsigned char 형 중 어느 형과 동일한지 여부를 결정하는 방법이 있다. 다음을 포함하는 프로그램을 작성하여라.

C code

```
#include <stdio.h>
int main (void) {
char c = -1;
signed char s=-1;
unsigned char u = -1;
printf ("c = %d s = %d u = %d ₩n",c,s,u);
return 0;   }
```

Result (결과)

```
[romance@161s ch3_code]$ ./a.out
c = -1 s = -1 u = 255
[romance@161s ch3_code]$
```

이것으로 char형은 singed char 형과 동일함을 알 수 있다.

2-11. 다음 코드를 실행하여 보고, 왜 가장 큰 정수 값을 출력하는 지를 설명하여라.

```
#include <stdio.h>
int main (void)
{
unsigned long val = -1;
printf ("The biggest number : %luWn",val);

return 0;
}
```

Sol) signed 의 경우 -1은 모든 bit가 1이다. 따라서
이것이 unsigned로 변환되면 가장 큰 정수값을 출력하게 되는 셈이다.

3-12. char형 변수는 작은 정수 값을 저장하는 데 사용 될 수 있다. 만일 큰 값이 char 형 변수에 저장되면 어떻게 되는가? 다음 코드를 살펴보자.

```
char c1 = 256,c2 = 257;
printf ("c1 = %dWn,c2 = %dWn",c1,c2);
```

Sol)

```
[romance@161s ch3_code]$ cc 3_12.c
3_12.c: In function `main':
3_12.c:4: warning: overflow in implicit constant conversion
3_12.c:4: warning: overflow in implicit constant conversion
[romance@161s ch3_code]$ ./a.out
c1 = 0
,c2 = 1
[romance@161s ch3_code]$
```

overflow가 일어 나기 때문에 char에 저정되는 값은 실제로 c1 % 255 가 될 것이다.

3-13. 다음 표는 대부분의 컴퓨터에서 기본형을 저장하는 데 필요한 바이트 수를 보여주고 있다. 사용자 컴퓨터에서의 값은 어떻게 되는가? 표를 완성하는 프로그램을 작성하고 실행해 보아라.

기본형	4바이트 워드 컴퓨터에서 요구되는 메모리	2 바이트 워드 컴퓨터에서 요구되는 메모리	사용자 컴퓨터에서 요구되는 메모리
char	1	1	1
short	2	2	2
int	4	2	4
unsigned	4	2	4
long	4	4	4
float	4	4	4
double	8	8	8
long double	?	?	12

```
#include <stdio.h>
int main (void)
{
printf ("%d₩n",sizeof(char));
printf ("%d₩n",sizeof(short));
printf ("%d₩n",sizeof(int));
printf ("%d₩n",sizeof(unsigned));

printf ("%d₩n",sizeof(long));
printf ("%d₩n",sizeof(float));
printf ("%d₩n",sizeof(double));
printf ("%d₩n",sizeof(long double));

return 0;
}
```

3-14. ‘A’ 같은 문자 상수의 형은 C에서는 int 형이고 C++에서는 char형이다. 다음 프로그램을 먼저 C로 컴파일 하고 그 다음 C++로 컴파일 하여라. 각 경우에 어떻게 출력 되는가?

```
#include <stdio.h>
int main (void)
{
printf("sizeof('A') = %u ₩n",sizeof('A'));
return 0;
}
```

a) C로 컴파일시

```
[romance@161s ch3_code]$ ./a.out
sizeof('A') = 4
[romance@161s ch3_code]$
```

b) C++로 컴파일시

```
[romance@161s ch3_code]$ ./a.out
sizeof('A') = 1
[romance@161s ch3_code]$
```

3-15. 먼저 다음 코드의 출력을 생각하여 적어보고, 프로그램을 작성하여 답을 확인해 보아라.

```
char c = 'A';
printf("sizeof(c) = %u₩n",sizeof(c));
printf("sizeof('A') = %u₩n",sizeof('A'));
printf("sizeof(c+ c) = %u₩n",sizeof(c+ c));
printf("sizeof(c = 'A') = %u₩n",sizeof(c = 'A'));
```

C++로 이 프로그램을 출력한다면, 출력결과는 어떻게 달라지겠는가?

Sol)

a) C로 출력했을 경우

```
[romance@161s ch3_code]$ ./a.out
sizeof(c) = 1
sizeof('A') = 4
sizeof(c+ c) = 4
sizeof(c = 'A') = 1
[romance@161s ch3_code]$
```

b) C++로 출력 했을 경우

```
[romance@161s ch3_code]$ ./a.out
sizeof(c) = 1
sizeof('A') = 1
sizeof(c+ c) = 4
sizeof(c = 'A') = 1
[romance@161s ch3_code]$
```

3-16. $N_{min_u_long}$과 $N_{max_u_long}$을 사용자 시스템에서 unsigned 형으로 저장할 수 있는 최소 및 최대 값을 나타낸다고 할 때, 이 값들은 어떻게 되겠는가?

Sol) limit.h 파일을 참조하면 된다.

3-17. 24시간 표현을 갖는 시계에서 0시는 밤 12시이고, 23시는 밤 11시이다. 이런 시계에서는 23에 1을 더하면 24가 아니라 0이다. 즉, 24는 존재하지 않는다. 마찬가지로 22에 5를 더하면 3이 된다. 이러한 모듈러 24 연산이라고 한다. 대부분의 컴퓨터에서는 모듈러 수행을 한다. 이를 확인하여라.

Sol)

C code

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

```
unsigned u = UINT_MAX;
for (i = 0; i < 10; ++i)
printf("%u +  %d = %u₩n",u,i,u+ i);
for (i= 0; i <10; ++i)
printf ("%u * %d = %u₩n",u,i,u*i);

return 0;
}
```

Result (결과)

```
[romance@161s ch3_code]$ ./a.out
4294967295 +  0 = 4294967295
4294967295 +  1 = 0
4294967295 +  2 = 1
4294967295 +  3 = 2
4294967295 +  4 = 3
4294967295 +  5 = 4
4294967295 +  6 = 5
4294967295 +  7 = 6
4294967295 +  8 = 7
4294967295 +  9 = 8
4294967295 * 0 = 0
4294967295 * 1 = 4294967295
4294967295 * 2 = 4294967294
4294967295 * 3 = 4294967293
4294967295 * 4 = 4294967292
4294967295 * 5 = 4294967291
4294967295 * 6 = 4294967290
4294967295 * 7 = 4294967289
4294967295 * 8 = 4294967288
4294967295 * 9 = 4294967287
[romance@161s ch3_code]$
```

3-18. ANSI C 는 이진 부동 소수점 산술식을 위한 IEEE 표준화 (ANSI/IEEE std 754-1985)의 권고를 따르도록 제안하고 있다. 사용자 시스템은 이러한 권고를 고 있는가? 부동형 변수에 해당 값의 범위를 초과하는 값을 저장할 때 어떤 일이 발생하는가를 검사해 보자. 다음 코드를 포함하는 프로그램을 작성하여라.

```
float x = 1e+307;
float y = x * x;
printf("x = %e y = %e₩n",x,y);
```

Sol)

```
[romance@161s ch3_code]$ ./a.out
x = inf y = inf
[romance@161s ch3_code]$
```

3-19. 수학에서 e와 π 값은 잘 알려져 있다. E는 자연로그의 기수이고, π 는 원주율이다. 이때 e^{π} 와 π^{e} 의 값 중 어느 것이 더 큰가?

Sol) e^{π} 이 더 크다.

3-20. sqrt() 함수의 인자로 음수를 사용하면 어떤 일이 발생하는가? 어떤 컴파일러는 sqrt(-1.0)과 같은 호출에 대해 실행시간 오류를 발생시키고, 어떤 컴파일러는 NaN라는 값을 리턴한다. 사용자 컴퓨터는 어떤 값을 리턴하는가?

Sol)

```
[romance@161s ch3_code]$ ./a.out
sqrt(-1.0) = nan
[romance@161s ch3_code]$
```

3-21. pow(x,x)를 호출할 때 x의 값이 너무 크면 실행시간 오류를 발생되거나, pow(x,x) 값을 printf()로 출력할 경우 Inf 또는 infinity가 출력된다. 확인하여라.

```
(생략)
```

3-22. 전통적인 C에서는 float은 double로 자동승격된다. ANSI C 에서는 수학수식에서 float을 double로 승격할 수 있지만, 요구사항은 아니다. 대부분의 ANSI C 컴파일러 는 수식에서 확장없이 float을 사용하는 것으로 조사되었다. 다음 코드를 사용하여 사용자 시스템에서는 어떻게 동작하는 지를 확인해 보아라.

```
float x = 1.0, y = 2.0;
printf("%s%u₩n%s%u₩n%s%u₩n","sizeof(float)",sizeof(float)
,"sizeof(double)",sizeof(double),"sizeof(x+ y)",sizeof(x+ y));
```

Result (결과)

```
[romance@161s ch3_code]$ ./a.out
sizeof(float)4
sizeof(double)8
sizeof(x+ y)4                    (승격되지 않았다.)
[romance@161s ch3_code]$
```

3-23. 전통적인 C에서, long float 형과 double 형은 동의어이다. 그러나, long,float은 타이핑하기 어려워서 일반화되지 못하였고, 자주 사용되지도 않았다. ANSI C에서는 long float 형이 제거 되었다. 그럼에도 불구하고 많은 ANSI C컴파일러에서 여전히 long float형을 사용할 수 있다. 사용자의 컴파일러에서도 이것이 가능한지를 조사하여라.

Sol) [romance@161s ch3_code]$ cc 3_23.c

```
3_23.c: In function `main':
3_23.c:4: long or short specified with floating type for `c'
3_23.c:4: the only valid combination is `long double'
[romance@161s ch3_code]$
```

사용 가능하지 않다.

3-24. 다음 코드를 포함하는 try_me 프로그램을 작성하여라.

```
#include <stdio.h>
int main (void)
{
int c;
while ( (c = getchar()) != EOF )
putchar(c);

return 0;
}
```

그리고 몇줄의 문장이 있는 infile이라는 파일 생성후 다음 실행시켜라.

Try_me <infile >outfile

이제 변수 c를 int형에서 char형으로 변경한 후 반복하여라.

Sol) 모두 infile에 있는 내용이 outfile로 복사 된다.

3-25. 수학 코드에서 fabs() 대신 abs()를 사용하면 이상한 결과를 얻을 것이다. 다음 프로그램을 수행해 보아라.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main (void)
{
double x = -2.357;
printf (" abx(x) = %e₩n",abs(x));
printf (" fabs(x) = %e ₩n",fabs(x));
return 0;
}
```

Sol)

```
[romance@161s ch3_code]$ ./a.out
 abx(x) = 2.644135e+00
 fabs(x) = 2.357000e+00
[romance@161s ch3_code]$
```

3-26. C에서 16진 정수로는 대문자, 소문자 또는 둘 다 사용할 수 있다. 다음 코드를 보자.

```
#include <stdio.h>
int main (void)
{ int a = 0xabc;
int b = 0xABc;
int c = 0xABC;
preintf ("a = %d b = %d c = %d",a,b,c);
return 0; }
```

Sol)

```
[romance@161s ch3_code]$ ./a.out
a = 2748 b = 2748 c = 2748
```

3-27. 다음 코드는 컴퓨터에 종속적이다. 예상 이외의 결과가 출력 된다면, 왜 그런지를 설명하여라.

```
#include <stdio.h>
int main (void)
{   char c = 0xff;
if ( c == 0xff)
   printf("Truth!₩n");
else
   printf("This needs to be explained!₩n");
return 0; }
```

Sol)

```
[romance@161s ch3_code]$ cc 3_27.c
3_27.c: In function `main':
3_27.c:5: warning: comparison is always false due to limited range of data type
[romance@161s ch3_code]$ ./a.out
This needs to be explained!
[romance@161s ch3_code]$
```

Chapter 3 End Point.