[컴구]김나현_2021120044

[프로그램 1]

```
#include <stdio.h>
int main()
{
    int n = 123456789;
    int nf, ng;
    float f;
    double g;
    f = (float)n;       // n=123456789
    g = (double)n;      // f = 123456792., n = 123456789
    nf = (int)f;        //f = 123456792., g = 123456789.00000
    ng = (int)g;        // f = 123456792., g=123456789.000000
    printf("nf=%d ng=%d\n", nf, ng);    // nf=123456792, ng =
    printf("%f, %f", f, g);
}
```

```
// 출력
nf=123456792 ng=123456789
```

int n → float f = (float) n → int nf = (int) f

먼저 int형 변수 n을 float형 변수 f로 변환하고, 이후에 다시 정수로 변환하여 int형 변수인 nf에 저장한다. float은 32비트로 표현되며, 정수 123456789를 float으로 형변환하면, float의 범위를 넘어가는 부분이 버려지게 된다. 123456789를 32비트 float로 표현하면 약 123456792에 가장 근사한 값이 되므로 f의 값이 123456792.000000이 된다.

따라서 float에서 int로 변환할 때, float형 변수 f의 값이 int형 변수 nf로 변환될 때, 소수점 이하의 값이 버려지고 정수 부분만 남게 되므로 nf에는 123456792가 저장되어 출력된다.

int n → double g = (double) n → int n = (int) g

마찬가지로, n을 double형 변수 g로 변환하고 다시 int형 변수인 ng로 변환하여 저장한다.

결과부터 말하자면 double 형변환은 float 형변환과 달리 비트 손실이 일어나지 않고 처음 저장한 123456789라는 값이 출력되었다. float 타입은 32비트를 사용하고, double 타입은 64비트를 사용하므로 더 큰 범위의 수를 저장하고 표현할 수 있다. 따라서 double로 형변환 하면서 소숫점까지 저장되었으나, int형 변수인 ng로 형변환을 해주었기 때문에 123456789가 출력된 것이다.

[프로그램 2]

```
#include <stdio.h>
int main ()
{
 double d;
 d = 1.0 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.
 printf ("d = %.20f\n", d);
}
```

```
// 출력
d = 2.00000000000000088818
```

인간이 계산할 때는 1.0에 0.1을 10번 더하게 될 경우, 2.0으로 값을 도출해낸다.

그러나 컴퓨터가 계산을 할 경우, 0.1은 2진수로 정확하게 연산되지 않는데 그 이유는 10진수에서의 0.1이 2진수로는 무한소수로 표현되기 때문이다. 따라서 부동소수점(floating point)의 한계로 인해 2.0에 근사한 2.00000000000000088818이라는 값이 출력된다.

[프로그램 3]

```
#include <stdio.h>
int main ()
{
 float f1 = (3.14 + 1e20) - 1e20;
 float f2 = 3.14 + (1e20 - 1e20);
 printf ("f1 = %f, f2 = %f\n", f1, f2);
}
```

```
// 출력
f1 = 0.000000, f2 = 3.140000
```

컴퓨터에서는 부동 소수점을 사용하여 실수를 표현한다. 부동소수점은 소수점 이하에 쓸 수 있는 비트 수가 한정되어 있어서, 매우 큰 수와 매우 작은 수를 함께 연산하게 될 경우 손실이 발생할 수 있다.

첫 번째 연산식을 살펴보면 (3.14 + 1e20) - 1e20 를 계산할 때 1e20은 1*10^20으로 3.14에 비해 비교적 매우 큰 수이다. 따라서 (3.14 + 1e20)을 먼저 계산하였을 때, 3.14는 매우 작은 수로 부동 소수점의 한계로 인해 연산의 결과에 미치는 영향이 미미하므로 사라진다고 볼 수 있다. 따라서 1e20 - 1e20을 계산하여 변수 f1의 값은 0에 가까운 수가 된다. 따라서 0.000000이 출력된다.

두 번째 연산식을 살펴보면 3.14 + (1e20 - 1e20)를 계산할 때, 먼저 (1e20 - 1e20)의 연산을 통해 0에 가까운 수가 나오게 되면 3.14 + 0 으로 3.140000을 출력하게 된다.

부동 소수점에서는 결합법칙을 지원하지 않기 때문에 결합법칙을 사용한 두 식은 서로 다른 결과를 출력한다.