HPFP 함수 식 설명

2023315171 장재원

```
int exponentF(int base, int exponent)
```
이 함수는 exponent가 양수일 때 쓰는 함수 이다. 따라서 결괏값은 base ^ (exponent)이다.

```
float exponentFloat(int base, int exponent)
```
이 함수는 exponent가 음수일 때 쓰는 함수이다. 따라서 결괏값은 base ^ (exponent)이다.

```
hpfp int_converter(int input)
```
이 함수는 정수를 hpfp로 바꾸는 함수이다.

```
    if (input == 1) {//예외 처리
        HPFP = 15360;
        return HPFP;
    }

    else if(input == -1){
        HPFP = 48128;
        return HPFP;
    }

    else if (input == 0){//예외 처리
        HPFP = 0;
        return 0;
    }
```
이 부분을 통해서 -1, 0, +1의 예외처리를 진행했다.

```
if (input < 0) {
        input = -input;
        s++;
    }//input 의 부호를 확인!
```
이 부분을 통해 input이 음수면 s를 증가시키고, input이 양수면 s는 0을 유지시킨다. 그리고 input은 양수로 바꾼다.

```
if(input > INTINF){ // 예외 처리
        input = INTINF;
    } // 초과하는 케이스에 대해 예외 처리
```
따라서 양수가 된 input이 만약 nan이면 예외처리를 한다.

```
while (inputCopy >= 2) {
        inputCopy /= 2;
        exponent++;
    }// input 의 크기를 확인한다.
```
frac부분을 구하기 위해 inputCopy를 2로 나누어서 inputCopy를 1과 2사이로 만들어 주고 그리고 나눈 2를 저장한다.

```
int index = 0;
    for (int i = exponent - 1; i >= 0; --i) {
        if (index == 10) break;
        frac[index] = input >> i & 1;
        index++;
    }//frac 를 대입! 맨앞에 있는 1 은 무시!
```

그리고 이 함수의 핵심파트이다. 시프트 연산자를 이용해서 알아서 2진수로 표현이 가능해지고, i의 초기값이 exponent – 1인 경우는 맨 앞에 있는 1을 무시하기 위함이다.

```
Exp1 = exponent + 15;
    exponentCopy = Exp1;
    while (exponentCopy >= 2) {
        exponentCopy /= 2;
        exponentNum++;
    }//똑같이 exponent 의 크기를 확인한다.

    index = 0;
    for (int i = exponentNum; i >= 0; --i) {
        Exp[index] = Exp1 >> i & 1;
        index++;
    }
```

이 부분은 위의 frac함수와 똑같이 exponent부분도 시프트연산자를 통해 담는 모습이다.

```
    HPFP = exponentF(2, 15) * s;
    for (int i = 0; i < 5; i++) {
        HPFP += Exp[i] * exponentF(2, 14 - i);
    }
    for (int i = 0; i < 10; i++) {
        HPFP += frac[i] * exponentF(2, 9 - i);
    }
    //HPFP 계산
    return HPFP;
}
```

따라서 배열에 들어있는 0과 1을 더해주면 완성이다.

```
int hpfp_to_int_converter(hpfp input)
```

이는 hpfp를 int로 바꾸는 함수이다.

```
    if(input == HPFPINF){//예외 처리
        return INT_MAX;
    }
    else if( exponentF(2, 16) > input && input > HPFPINF ){ // +NAN
        return INT_MIN;
    }//예외 처리
    else if( exponentF(2, 17) > input && input >= HPFPNEGINF){ // -NAN
        return INT_MIN;
```

```
    }
```
Input의 예외 처리를 전부 해준다.

```
    int s = input / exponentF(2, 15);
    input %= exponentF(2, 15); // 부호를 제외한 나머지
    int Exp = input / exponentF(2, 10);  // Exp 는 11 ~ 15 자리 사이에 있다.
    int E = Exp - 15; // E = Exp - Bias 이다.
    input %= exponentF(2, 10); //frac 부분을 계산한다. 256
    float answer = 1 + ((float)input / exponentF(2, 10));
    answer *= exponentF(2, E);
    if (s == 1) answer = -answer; // 부호만 계산하면 되니 맨 뒤에다가 놓자.
    return answer;
}
```
이는 hpfp가 s 1자리/ exponent 5자리/ frac부분 10자리를 만들어서 표현되는 것을 우리가 int로 전환하는 것뿐이다.

```
hpfp float_converter(float input) {
```
이는 float를 hpfp로 바꾸는 함수이다.

```
    if (input == 1) {//예외 처리
        return 15360;
    }

    else if(input == -1){
        return 48128;
    }

    else if (input == 0){//예외 처리
        return 0;
    }
```
이는 input이 -1/0/1일때의 예외처리를 하는 것이다.

```
    int s = 0;
    if (input < 0) {
        input = -input;
        s++;
    }//음수인지 아닌지를 체크!
```
이는 float가 음수이면 양수로 바꾸고, s를 1증가시키는 함수이다. 따라서 input은 양수로 유지가 된다,

```
    if(input > INTINF) // 예외 처리
        input = INTINF;
     // 초과하는 케이스에 대해 예외 처리
```
따라서 추가적인 예외처리를 해주면 된다.

```
    if(input > exponentFloat(2, -15)){ // Special case와 Normalized value일때
        int bigExp = 0;
        if(input > 1){
            for (int i = 16; i >= 0; i--) {
                if (input >= exponentF(2, i)) {
                    bigExp = i;
                    input /= exponentF(2, bigExp);
                    break;
                }
            }
        }//input > 1 일때
        else{
            for(int i = 0; i >= -14; i--){
                if(input >= exponentFloat(2, i)){
                    bigExp = i;
                    input /= exponentFloat(2, bigExp);
                    break;
                }
            }
        }//input < 1 일때
```

이 함수는 input을 2로 나누거나 곱하였을 때 input을 1과 2사이로 만들고 지수를 빼내는 작업을 하는 것이다.

```
        int E = bigExp + 15;
        int Exp[5] = { 0 };
        int index1 = 0;
        for (int i = 4; i >= 0; i--) {
            if (E >= exponentF(2, i)){
                Exp[index1] = 1;
                E -= exponentF(2, i);
            }
            index1++;
        }// E를 이진수로 표현하여 배열에다가 담았다.
        input -= 1;
        int frac[10] = { 0 };
        int index2 = 0;
        for (int i = -1; i >= - 10; i--) {
            if (input >= exponentFloat(2, i)){
                frac[index2] = 1;
                input -= exponentFloat(2, i);
            }
            index2++;
        }//frac 부분을 계산했다.
```

따라서 normalize 부분의 식을 따라가서 frac부분에 값을 담고 Exp부분에도 값을 담는 것이다.

```
        hpfp HPFP = 0;
        HPFP = exponentF(2, 15) * s;
        for (int i = 0; i < 5; i++) {
```

```
            HPFP += Exp[i] * exponentF(2, 14 - i);
        }
        for (int i = 0; i < 10; i++) {
            HPFP += frac[i] * exponentF(2, 9 - i);
        }
        //HPFP 계산
        return HPFP;
    }
```

나머지는 계산을 하는 것이다.

```
    else{ //Denormalize value 일때
        input *= exponentF(2, 14);
        int frac[10] = { 0 };
        int index2 = 0;
        for (int i = -1; i >= - 10; i--) {
            if (input >= exponentFloat(2, i)){
                frac[index2] = 1;
                input -= exponentFloat(2, i);
            }
            index2++;
        }//frac 부분을 계산했다.
        hpfp HPFP = 0;
        HPFP = exponentF(2, 15) * s;
        for (int i = 0; i < 10; i++) {
            HPFP += frac[i] * exponentF(2, 9 - i);
        }
        //HPFP 계산
        return HPFP;

    }
```

이 부분은 denormalize부분을 위한 함수이고 큰 틀은 비슷하다.

```
float hpfp_to_float_converter(hpfp input)
```

hpfp -> float이다.

```
    int s = input / exponentF(2, 15);
    input %= exponentF(2, 15); // 부호를 제외한 나머지
    int Exp = input / exponentF(2, 10);  // Exp 는 11 ~ 15 자리 사이에 있다.
```

hpfp에서 1자리 s/ 5자리 Exp/ 10자리 frac이다. 따라서 s와 Exp에 적절히 담아준다.

```
    if(Exp != 0){//Normalized value
        int E = Exp - 15; // E = Exp - Bias 이다.
        float frac = input % exponentF(2, 10); //frac 부분을 계산한다.
        float fracFull = 1 + frac / exponentF(2, 10);
```

```
        float result = 0;
        if(E >= 0)
        result = exponentF(2, E) * fracFull;
        else
        result = exponentFloat(2, E) * fracFull;
        if (s == 1) result = -result;
        return result;
    }
```

Exp !=0일 때 normalized value이니 이때 normalized value의 계산 식을 따르면 된다.

```
    else{//Denormalized value
        int E = -14;
        float frac = input % exponentF(2, 10); //frac 부분을 계산한다.
        float result = exponentFloat(2, E) * ((frac) / exponentF(2, 10));
        if (s == 1) result = -result;
        return result;
    }
}
```

이 부분은 denormalized value이다.

따라서 denormalized value의 계산 식을 따르면 된다.

```
hpfp addition_function(hpfp a, hpfp b)
```

이는 두개의 hpfp를 더하는 것이다.

```
    if(HPFPNAN>= a && a > HPFPINF) return HPFPNAN;
    if(a > HPFPNEGINF) return HPFPNAN;
    if(HPFPNAN>= b && b > HPFPINF) return HPFPNAN;
    if(b > HPFPNEGINF) return HPFPNAN;
    // NAN 의 범위는 모두 삭제
    if(a == HPFPINF){
        if(b == HPFPNEGINF)
            return HPFPNAN;
        else
            return HPFPINF;
    }
    else if(a == HPFPNEGINF){
        if(b == HPFPINF)
            return HPFPNAN;
        else
            return HPFPNEGINF;
    }
    //word 에서 나타낸 예외케이스는 모두 끝!
```

예외 케이스들은 모두 제외한다.

```
    int expA = (a % exponentF(2, 15)) / exponentF(2, 10);
    int expB = (b % exponentF(2, 15)) / exponentF(2, 10);
```

exp들을 먼저 계산해서 denormal + denormal인지 normal + denormal인지 normal + normal인지를 구분한다.

```
    // 1) Denorm + Denorm 일때
    if(expA == 0 && expB ==0){
        int sA = a / exponentF(2, 15);
        int sB = b / exponentF(2, 15);
        float mA = (float)(a % exponentF(2, 10)) / exponentF(2, 10);
        float mB = (float)(b % exponentF(2, 10)) / exponentF(2, 10);
        float m = exponentF(-1, sA) * mA + exponentF(-1, sB) * mB;
        int sM = 0;
        if(m < 0){
            sM = 1;
            m = -m;
        }
        if(m >= 1){ // Denorm + Denorm = norm 일때
            int Exp = 1;
            hpfp HPFP = 0;
            m -= 1;
            HPFP += (exponentF(2, 15) * sM) + (Exp * exponentF(2, 10)) + (m *
exponentF(2, 10));
            return HPFP;
        }
        else{ // Denorm + Denorm = Denorm 일때
            hpfp HPFP = 0;
            HPFP += (exponentF(2, 15) * sM) + (m * exponentF(2, 10));
            return HPFP;
        }
    }
```

둘 다 denorm일때이다. 따라서 denorm의 계산식을 따라 s , exp, frac부분을 발굴해낸다.

그리고 더하기를 할때에는 e를 같게 고정시키고 나머지 부분을 m에다가 몰아넣는 식으로 진행한다. 근데 이 부분은 e = -14로 고정이 되어있으니 그냥 더하면 된다.

그리고 denorm + denorm은 norm이 될수도, denorm이 될 수도 있으니 이를 유의해서 계산을 하면 된다.

```
else if(expA == 0 || expB == 0){
        if(expB == 0){//A 가 denormal 이고, B 가 normalize 인경우
            hpfp temp = 0;
            temp = a;
            a = b;
            b = temp;
        }
        int sA = a / exponentF(2, 15);
```

```
        int sB = b / exponentF(2, 15);
        float mA = (float)(a % exponentF(2, 10)) / exponentF(2, 10);
        float mB = 1 + ((float)(b % exponentF(2, 10)) / exponentF(2, 10));
        int eA = -14;
        int eB = ((b % exponentF(2, 15)) / exponentF(2, 10)) - 15;
        float m = mA * exponentF(-1, sA) + mB * exponentF(2, eB + 14) *
exponentF(-1, sB);
        int sM = 0;
        if(m < 0){ // m이 음수 일때
            sM++;
            m = -m;
        }
        // denormal + normal == normal인 경우
        if(m >= 1){
            int e = 0;
            while(m >= 2){
                m /= 2;
                e++;
            }
            int Exp = e - 14 + 15;
            hpfp HPFP = (exponentF(2, 15) * sM) + (exponentF(2, 10) * Exp) +
((m - 1) * exponentF(2, 10));
        }

        else{ // denormal + normal == denormal인 경우
            hpfp HPFP = (exponentF(2, 15) * sM) + (m * exponentF(2, 10));
        }
    }
```

이 부분은 denormal + normal인 경우이다. Denormal인 경우 e = -14이고,. Normal은 -14이상이니, 따라서 e를 -14로 고정한 상태에서 나머지 부분을 묶어서 더한 다음에 다시 2로 나누거나 2를 곱해서 e에 추가적인 부분을 더하면 된다.

```
    else{
        int aRemainder = a % exponentF(2, 15);
        int bRemainder = b % exponentF(2, 15);
        if(aRemainder < bRemainder){//a와 b의 절댓값중 더 큰 값을 a에, 더 작은
값을 b에 두기 위함.
            hpfp temp = a;
            a = b;
            b = temp;
        }
        int sA = a / exponentF(2, 15);
        int sB = b / exponentF(2, 15);
        int eA = (a % exponentF(2, 15)) / exponentF(2, 10) - 15;
        int eB = (b % exponentF(2, 15)) / exponentF(2, 10) - 15;
        float mA = 1 + ((float)(a % exponentF(2, 10))) / exponentF(2, 10);
        float mB = 1 + ((float)(b % exponentF(2, 10))) / exponentF(2, 10);
```

```
        float m = mA * exponentF(-1, sA) * exponentF(2, eA - eB) + mB *
exponentF(-1, sB);
        int sM = 0;
        if(m < 0){
            m = -m;
            sM++;
        }
        //m >= 2
        int exponent = 0;
        if(m >= 2){
            while(m >= 2){
                m /= 2;
                exponent++;
            }
        }
        //0 < m < 1
        if(m < 1){
            while(m < 1){
                m *= 2;
                exponent--;
            }
        }

        int realExponent = eB + exponent;
        if(realExponent <= -16){//Denormalized value
            m *= exponentFloat(2, 14 + realExponent);
            hpfp HPFP = exponentF(2, 15) * sM + m * exponentF(2, 10);
            return HPFP;
        }
        else if(realExponent >= 16){//special value
            return HPFPINF;
        }
        else{//normalize value
            hpfp HPFP = (exponentF(2, 15) * sM) + (exponentF(2, 10) *
(realExponent + 15)) + ((m - 1) * exponentF(2, 10));
            return HPFP;
        }
    }
```

이 부분은 norm + norm이다. Norm + norm은 denormal이 될 수도, normal이 될수도, special case가 될 수도 있다. 따라서 계산된 값의 exp를 확인하여 이를 구분하고 알맞은 부분을 찾아가면 된다.

```
hpfp multiply_function(hpfp a, hpfp b){
```
이 함수는 hpfp인 두 매개변수를 곱하는 것이다.

```
    if(HPFPNAN>= a && a > HPFPINF) return HPFPNAN;
    if(a > HPFPNEGINF) return HPFPNAN;
    if(HPFPNAN>= b && b > HPFPINF) return HPFPNAN;
    if(b > HPFPNEGINF) return HPFPNAN;
    if(a == HPFPINF){
        if(b > 0) return HPFPINF;
        else if(b == 0) return HPFPNAN;
        else return HPFPNEGINF;
    }
    else if(a == HPFPNEGINF){
        if(b > 0) return HPFPNEGINF;
        else if(b == 0) return HPFPNAN;
        else return HPFPINF;
    } // 입력한 값에 대한 예외 처리 끝
```
입력한 값에 대해 예외처리를 한다.

```
    int expA = (a % exponentF(2, 15)) / exponentF(2, 10);
    int expB = (b % exponentF(2, 15)) / exponentF(2, 10);
```
exp들을 먼저 계산해서 denormal * denormal인지 normal * denormal인지 normal * normal인지를 구분한다.

곱하기의 핵심은 e 부분들은 더하고, m부분들은 서로 곱한다는 것이다.

따라서 e들을 서로 더한 결괏값을 비교하여 이를 norm denorm special 3가지를 구분해 볼 수 있겠다,

```
    if(expA == 0 && expB == 0)// Denorm * Denorm 은 hpfp 로 담지 못한다.
        return 0;
```
denorm * denorm은 값이 hpfp가 표현하기에 너무 작다.

```
else if(expA == 0 || expB == 0){ //Denorm * norm 인 경우
        if(expB == 0){//B 가 denormal 인 경우
            hpfp temp = a;
            a = b;
            b = temp;
        }//무조건 a 가 denormal 이 되게 세팅을 해 놓자.

        int sA = a / exponentF(2, 15);
        int sB = b / exponentF(2, 15);
        int eA = -14;
        int eB = (b % exponentF(2, 15)) / exponentF(2, 10) - 15;
        float mA = (float)(a % exponentF(2, 10)) / exponentF(2, 10);
        float mB = 1 + (float)(b % exponentF(2, 10)) / exponentF(2, 10);
```

```
        int eM = eA + eB;
        int sM = sA + sB;
        if(sM == 2)
            sM = 0;
        float m = mA * mB;
        float check = 0;
        if(eM >= 0)
            check = exponentF(2, eM) * m;
        else
            check = exponentFloat(2, eM) * m;
        if(check < exponentFloat(2, -15)){//곱하기의 결과가 denorm 일때 // m 의
값은 무조건 1 보다 작다.
            if(eM >= -14)
                m *= exponentF(2, 14 + eM);
            else
                m *= exponentFloat(2, 14+ eM);
            eM = -14;
            m *= exponentF(2, 10);
            hpfp mR = (int)m;
            float mF = m - mR;
            //round to even 을 생각해보자.
            if(mF > 0.5)
                mR++;
            else if(mF == 0.5){
                if((mR % 2) != 0)//mR 이 홀수일때는 1 증가!
                    mR++;
            }
            hpfp HPFP = exponentF(2, 15) * sM + mR;
            return HPFP;
        }
        else{//곱하기의 결과가 norm 일때
            int exponent = 0;
            if(m < 1){
                while(m < 1){
                    m *= 2;
                    exponent--;
                }
            }
            eM += exponent;
            m -= 1;
            m *= exponentF(2, 10);
            hpfp mR = (int)m;
            float mF = m - mR;
            //round to even 을 생각해보자.
            if(mF > 0.5)
                mR++;
            else if(mF == 0.5){
                if((mR % 2) != 0)//mR 이 홀수일때는 1 증가!
```

```
                    mR++;
            }
            hpfp HPFP = exponentF(2, 15) * sM +((eM + 15) * exponentF(2, 10))+
mR;
            return HPFP;
        }

    }
```

이 부분은 denorm * norm이다. Denorm * norm부분은 norm이 될수도, denorm이 될 수도 있다.

그리고 각자의 m을 곱한 m을 정수부(mR)와 소수부(mF)로 찢는다. 그래서 round-to-even을 통해 mF가 0.5보다 크면 1증가 0.5보다 작으면 그대로, 0.5이면 mR이 짝수면 1증가, 홀수면 그대로를 유지한다.

```
    else{ //norm * norm 인 경우
        int sA = a / exponentF(2, 15);
        int sB = b / exponentF(2, 15);
        int eA = (a % exponentF(2, 15)) / exponentF(2, 10) - 15;
        int eB = (b % exponentF(2, 15)) / exponentF(2, 10) - 15;
        float mA = 1 + (float)(a % exponentF(2, 10)) / exponentF(2, 10);
        float mB = 1 + (float)(b % exponentF(2, 10)) / exponentF(2, 10);
        int eM = eA + eB;
        int sM = sA + sB;
        if(sM == 2)
            sM = 0;
        float m = mA * mB;
        float check = 0;
        if(eM >= 0)
            check = m * exponentF(2, eM);
        else
            check = m * exponentFloat(2, eM);
        if(check > exponentF(2, 16)){// norm * norm = specialized value 일때
            return HPFPINF;
        }
        else if(check < exponentFloat(2, -15)){// norm * norm = denormalize
value 일때
            if(eM >= -14)
                m *= exponentF(2, 14 + eM);
            else
                m *= exponentFloat(2, 14+ eM);
            eM = -14;
            m *= exponentF(2, 10);
            hpfp mR = (int)m;
            float mF = m - mR;
            //round to even 을 생각해보자.
            if(mF > 0.5)
                mR++;
```

```
            else if(mF == 0.5){
                if((mR % 2) != 0)//mR이 홀수일때는 1증가!
                    mR++;
            }
            hpfp HPFP = exponentF(2, 15) * sM + mR;
            return HPFP;
        }
        else{//check가 normal일때
            int exponent = 0;
            if(m >= 2){
                while(m >= 2){
                    exponent++;
                    m /= 2;
                }
            }
            eM += exponent;
            m -= 1;
            m *= exponentF(2, 10);
            hpfp mR = (int)m;
            float mF = m - mR;
            //round to even을 생각해보자.
            if(mF > 0.5)
                mR++;
            else if(mF == 0.5){
                if((mR % 2) != 0)//mR이 홀수일때는 1증가!
                    mR++;
            }
            hpfp HPFP = (exponentF(2, 15) * sM) +((eM + 15) * exponentF(2,
10))+ mR;
            return HPFP;
        }
    }
}
```

Norm * norm 인 경우이다.

```
char* comparison_function(hpfp a, hpfp b){
```

comparison은 a와 b를 비교하는 것이다.

```
    int signA = a / exponentF(2,15);
    int signB = b/ exponentF(2,15);
    if(signA == 1){
        if(signB == 0){
            *pointer = '<';
        }
```

```
    }
    else{//a = 0
        if(signB == 1){
            *pointer = '>';
        }
    }
```

부호를 통해서 간단히 비교가 가능하다.

```
    if(exponentF(2, 15)> a && a > HPFPINF)
        *pointer = '=';
    else if(a > HPFPNEGINF)
        *pointer = '=';

    if(exponentF(2, 15)> b && b > HPFPINF)
        *pointer = '=';
    else if(b > HPFPNEGINF)
        *pointer = '=';
```

예외케이스를 처리하자.

이젠 이 함수의 가장 중요한 부분이 나온다. Hpfp는 s/exp/frac로 나누어져있다. 따라서 exp의 차이만큼을 한쪽 frac에다가 넣고 비교를 하면 간단히 된다.

```
    //a 와 b 의 부호가 같다.
    if(a == 1){ // a 와 b 가 모두 음수일때
        if(subE >= 0){
            int fracA = a % exponentF(2,10);
            int fracB = b % exponentF(2,10);
            fracA *= exponentF(2, subE);
            if(fracA > fracB){
                *pointer = '<';
            }
            else if (fracA == fracB){
                *pointer = '=';
            }
            else{
                *pointer = '>';
            }
        }
        else{ // a 와 b 가 모두 음수일때
            int fracA = a % exponentF(2,10);
            int fracB = b % exponentF(2,10);
            fracA *= exponentFloat(2,subE);
            if(fracA > fracB){
                *pointer = '<';
            }
            else if (fracA == fracB){
                *pointer = '=';
            }
```

```
            else{
                *pointer = '>';
            }
        }
    }


    else{ // a와 b가 모두 양수일때
        if(subE >= 0){
            int fracA = a % exponentF(2,10);
            int fracB = b % exponentF(2,10);
            if(fracA * exponentF(2,subE) > fracB){
                *pointer = '>';
            }
            else if (fracA * exponentF(2,subE) == fracB){
                *pointer = '=';
            }
            else{
                *pointer = '<';
            }
        }
        else{
            int fracA = a % exponentF(2,10);
            int fracB = b % exponentF(2,10);
            if(fracA * exponentFloat(2,subE) > fracB){
                *pointer = '>';
            }
            else if (fracA * exponentFloat(2,subE) == fracB){
                *pointer = '=';
            }
            else{
                *pointer = '<';
            }
        }
    }
    return pointer;
}
```

이렇게 하면 된다.

```
char* hpfp_to_bits_converter(hpfp result)
```

hpfp를 비트마다 쪼개서 배열에다가 담으면 된다.

```
    char* bitString = (char*) malloc(17 * sizeof(char));
    for (int i = 0; i < 16; i++) {
        bitString[i] = ((result / exponentF(2, 15 - i)) % 2) + '0';
    }
    bitString[16] = '\0';
    return bitString;
```

```
}
```

```
char* hpfp_flipper(char* input){
```

이는 hpfp를 float로, float를 hpfp로 바꾸면 된다.

```
    hpfp numHPFP = 0;
    for(int i = 0; i <= 15; i++){
        numHPFP += (input[i] - '0') * exponentF(2, 15 - i);
    }//hpfp 완성
    float num = hpfp_to_float_converter(numHPFP);//hpfp를 float로 바꾼다.
    int exponent = 0;
    //int와 float가 같을 때
    for(int i = 0; i <= 10; i++){
        if((int) num == (float) num)
            break;
        exponent++;
        num *= 10;
    }
```

이는 hpfp가 만약 소수형식으로 주어진다면 이를 정수로 만들기 위해 10을 계속 곱해서 만드는 것이다.

```
    float numCopy = num;
    if(numCopy < 0)
        numCopy = -numCopy;
    int numLong = 0; // numLong은 num의 자릿수
    for(int i = 0; i <= 30; i++){
        if(numCopy < 1)
            break;
        numCopy /= 10;
        numLong++;
    }
```

총 자리수를 구하자.

```
    //filpper 시작
    float answer = 0;
    for(int i = 1; i <= numLong; i++){
        answer += ((int)(num / exponentF(10, i - 1)) % 10) * exponentF(10,
numLong - i);
    }

    answer /= exponentF(10, exponent);
    // filpper 완료

    hpfp answerHpfp = float_converter(answer);
    return hpfp_to_bits_converter(answerHpfp);
```

```
}
```

Filpper를 하면 된다.