

Los números reales en coma flotante se convertían a binario en tres pasos:

1. Convertir al sistema binario
2. Escribir en notación científica
3. Seguir el standard IEEE754 para 32 bits

Por una parte la parte entera del número real se convertía a binario y por otra la parte fraccionaria, según el algoritmo que se explicaba en el vídeo <https://www.youtube.com/watch?v=VMcypTxcbyY>. Este algoritmo debería haber sido el utilizado, no permitiéndose el uso de otros algoritmos.

En la práctica anterior, hemos terminado de convertir un número real en binario, según el standard IEEE754.

En esta práctica se generarán repetidamente, y de manera aleatoria, números reales en base decimal, que deberán ser convertidos a binario, recalculando finalmente el número real correspondiente al binario. La cantidad de números reales estará determinada por `#define B`, p.ej. `#define B 4`.

```
real random:
  14319.458008
numero real convertido a binario:
  11011111101111.0111010101
real recalculado:
  14319.458008

real random:
  9024.245117
numero real convertido a binario:
  10001101000000.0011111011
real recalculado:
  9024.245117

real random:
  11906.300781
numero real convertido a binario:
  10111010000010.01001101
real recalculado:
  11906.300781

real random:
  1761.237915
numero real convertido a binario:
  11011100001.0011110011101
real recalculado:
  1761.237915
```

Figura 1. Ejemplo de ejecución del programa

Los números **aleatorios** se generarán usando funciones predefinidas, de bibliotecas, de la siguiente manera:

```
#include "time.h" // time()
#include "stdlib.h" // srand(), rand()

//...
srand((int)time(NULL) );
aleatorio= rand(); // rand()%3 genera el aleatorio 0, 1 ó 2
//...
```

Se usará además, para el cálculo de la potencia, la función predefinida `pow()`:

```
#include "math.h"
```

```
// float pow(float,float)
```

Se reutilizarán las funciones necesarias definidas en la práctica anterior, y además se definirán, y **se usarán, todas y cada una** de las funciones cuyos prototipos se dan en el siguiente recuadro.

Se añade también la restricción de que la función *longitud* se implementará de forma **recursiva**, no de forma iterativa.

```
// defines
```

```
#define maximo_chars 64
```

```
#define B 4
```

```
// prototipos de las funciones que se deben definir en esta práctica
```

```
int longitud(int ); // función recursiva
```

```
// longitud(543)=3
```

```
float random_real();
```

```
// devuelve un número real, calculado aleatoriamente
```

```
void real_binario(float real,char bits_binario[maximo_chars]);
```

```
// convierte un numero real en base decimal a base binaria, printando el binario
```

```
float numero_real(char b[maximo_chars]){
```

```
// devuelve en base 10 el numero real correspondiente a un binario con parte entera y fraccionaria
```

```
int main() {
```

```
    srand((int)time(NULL) );
```

```
    char bits[maximo_chars];
```

```
    int i;
```

```
    for(i=0; i<B; i++){
```

```
        float real = random_real();
```

```
        printf("real random:\n\t%f\n", real);
```

```
        printf("numero real convertido a binario:");
```

```
        real_binario(real, bits);
```

```
        printf("\nreal recalculado:");
```

```
        printf("\n\t%f", numero_real(bits));
```

```
        printf("\n\n\n");
```

```
    }
```

```
    return 0;
```

```
}
```

// funciones definidas en prácticas anteriores y usadas en ésta

```
void convertir_entero(int x,char array[maximo_chars]){
    int bit,i=maximo_chars-1;
    do{
        if(((float)x/2-x/2)>= 0.5){
            bit=1;
        }else{
            bit=0;
        }
        array[i]=bit+48;
        i--;
        x/=2;
    }while(x!=0);
}

void convertir_fraccionario(float x,char array[maximo_chars]){
    float f = x;
    insertar_final(array,'.');
    do{
        if(f*2>=1){
            insertar_final(array,'1');
            f=(f*2)-1;
        }else{
            insertar_final(array,'0');
            f=(f*2);
        }
    }while(f!=0.0);
}

void resetear(char binario[maximo_chars]){
    for(int i = 0;i<maximo_chars;i++){
        binario[i] = ' ';
    }
}

void mover_izda(char array [maximo_chars]){
    int i=bits_blanco(array);
    for(i;i<maximo_chars;i++){
        array[i-1]=array[i];
    }
    array[maximo_chars-1] = ' ';
}

int bits_blanco(char array[maximo_chars]){
    int i = 0;
    while(array[i]!=' '){
        i++;
    }
    return i;
}

void insertar_final(char array[maximo_chars],char c){
    mover_izda(array);
    array[maximo_chars-1]=c;
}
```

```
void printar_binario(char array[maximo_chars]){
    int i = bits_blanco(array);
    printf("\n\t");
    for(i<maximo_chars;i++){
        printf("%c",array[i]);
    }
}

int posicion_punto_decimal (char array[maximo_chars]){
    int encontrado=0,i=maximo_chars-1;
    while(encontrado==0 && i<maximo_chars){
        if(array[i] == '.'){
            encontrado = 1;
        }else{
            i--;
        }
    }
    return i;
}

void binario_entera(int num, char array[maximo_chars]){
    int i=maximo_chars-1, bit;
    do{
        bit = (int)num%2;
        array[i] = bit+48;
        num /= 2;
        i--;
    } while (num>=1);
}

void binario_fraccionaria(float num, char array[maximo_chars]){
    insertar_final(array, '.');
    do{
        if((int)(num*2)!=0){
            insertar_final(array, '1');
        } else {
            insertar_final(array, '0');
        }
        num = num *2 - (int)(num*2);
    } while (num != 0.0);
}

// funciones que se deben definir en esta práctica

int longitud(int num){
    if(num == 0){
        return 0;
    } else {
        return 1 + longitud(num/10);
    }
}

float random_real(){
    int signo = rand()%2;
    float decimal = (float)rand();
    decimal = decimal/pow(10, longitud((int)decimal)) + (float)rand();
}
```

```
    if(signo == 1){
        decimal = (decimal) * (-1);
    }
    return decimal;
}

void real_binario(float real, char bits_binario[maximo_chars]){
    char sig = '+';
    if(real < 0){
        sig = '-'; real *= (-1);
    }
    int x = (int)real;
    float f = real - x;
    resetear(bits_binario);
    convertir_entero(x, bits_binario);
    convertir_fraccionario(f, bits_binario);
    int i = bits_blanco(bits_binario) - 1;
    bits_binario[i] = sig;
    printar_binario(bits_binario);
}

float numero_real(char a[maximo_chars]){
    int pospunto = posicion_punto_decimal(a);
    int bitsblanco = bits_blanco(a);
    int p = pospunto - 1;
    int n = pospunto + 1;
    float numeroE = 0, i = 0;
    while(p >= 0){
        if(a[p] == '1'){
            numeroE = numeroE + pow(2,i);
        }
        i++; p--;
    }
    i = -1;
    while(n < maximo_chars){
        if(a[n] == '1'){
            numeroE = numeroE + pow(2,i);
        }
        n++; i--;
    }
    if(a[bitsblanco] == '-'){
        numeroE *= (-1);
    }
    return numeroE;
}
```