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 conviertía a binario y por otra la parte fraccionaria, según el algoritmo que se explicaba en el vídeo https://www.youtube.com/watch?v=VMcypTxcbvY. 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.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define maxChars 64
#define B 4

//Prototipos

//Practica 7

void resetear(char []);
// se resetea el array

void binarioEntera(int, char []);
// un int se convierte a binario (se almacena en el array)

void binarioFraccionaria(float , char []);
// un float se convierte a binario (se almacena en el array)

void insertarFinal(char [],char );
// se inserta un char al final del array, desplazando el resto a la izquierda

void printBinario(char []);
// se printa el array con los char del numero binario

int bitsBlanco(char []);
// chars en blanco en el array

int posicionPuntoDecimal (char []);
// posicion en el array de '.'

//Practica 8
```

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

float randomReal();
// devuelve un número real, calculado aleatoriamente

void realBinario(float ,char []);
// convierte un numero real en base decimal a base binaria, printando el
binario

float numeroReal(char []);
// devuelve en base 10 el numero real correspondiente a un binario con parte
entera y fraccionaria


int main(){
    char binary[maxChars];
    char numReal[maxChars];
    float real = 0.0;
    float realAux = 0.0;

    srand(time(NULL));

    for(int i = 0; i < B; i++){
        resetear(binary);
        resetear(numReal);
        real = randomReal();
        printf("\nNumero random: %.2f\n", real);
        printf("Numero real %.2f convertido a binario:\n", real);
        realBinario(real, binary);
        printf("\nNumero real recalculado:\n%.2f\n\n", real);
        realAux = numeroReal(numReal);
    }

    return 0;
}

//Definicion de funciones
//Practica 7

void resetear(char binary[]){
    for (int i = 0; i < maxChars; i++){
        binary[i] = ' ';
```

```
    }
}

void binarioEntera(int num, char binary[]){
    int i = maxChars - 1;

    while (num > 0){
        if(10 * (((float) num / 2) - (num / 2)) >= 5){
            binary[i--] = '1';
        } else{
            binary[i--] = '0';
        }
        num /= 2;
    }
}

void binarioFraccionaria(float decimal, char binary[]){
    insertarFinal(binary, '.');
    while(decimal != 0.0){
        if((int) (decimal * 2) != 0){
            insertarFinal(binary, '1');
        } else{
            insertarFinal(binary, '0');
        }
        decimal = decimal * 2 - ((int) (decimal * 2));
    }
}

void insertarFinal(char cad[], char car){
    for (int i = 0; i < maxChars - 1; i++){
        cad[i] = cad[i + 1];
    }
    cad[maxChars - 1] = car;
}

void printBinario(char binary[]){
    for(int i = 0; i <= maxChars; i++){
        if(binary[i] != ' '){
            printf("%c", binary[i]);
        }
    }
}

int bitsBlanco(char cad[]){
```

```
    int count = 0;

    for(int i = 0; i < maxChars; i++){
        if(cad[i] == ' '){
            count++;
        }
    }
    return count;
}

int posicionPuntoDecimal (char cadena[]){
    int posicion = 0;

    for(int i = 0; i < maxChars; i++){
        if(cadena[i] == '.'){
            posicion = i;
        }
    }

    return posicion;
}

//Practica 8

float randomReal(int rango){
    float real = 0.0;
    float parteDecimal = 0.0;
    int parteEntera = 0;

    parteEntera = rand();
    parteDecimal = rand();
    real = parteEntera + (parteDecimal / pow(10, longitud(parteDecimal)));

    return real;
}

int longitud(int num){
    if(num < 10){
        return 1;
    } else{
        return 1 + longitud(num / 10);
    }
}
```

```
void realBinario(float real,char bitsBinario[maxChars]){
    int parteEntera = (int) real;
    float decimal = real - (int) real;

    if(real < 0){
        parteEntera *= -1;
        binarioEntera(parteEntera, bitsBinario);
        decimal *= -1;
        binarioFraccionaria(decimal, bitsBinario);
    } else {
        binarioEntera(parteEntera, bitsBinario);
        binarioFraccionaria(decimal, bitsBinario);
    }
    printBinario(bitsBinario);
}

float numeroReal(char binary[maxChars]){
    int i, j, t, p;
    float num = 0.0;

    //Parte entera
    for(i = t = bitsBlanco(binary), j = 0; i < (p =
posicionPuntoDecimal(binary)); i++, j++){
        if(binary[i] == '1'){
            num += pow(2, ((p - t) - 1) - j);
        }
    }

    //parte fraccionaria
    for(p = -1, i = (posicionPuntoDecimal(binary) + 1); i < maxChars; i++, p--
){
        if(binary[i] == '1'){
            num += pow(2, p);
        }
    }
    return num;
}
```