EJERCICIO 1

El objetivo del presente ejercicio es desarrollar un programa que permita averiguar cual es el numero mas pequeño que puede ser sumado con uno sin que se pierda información.

El programa realizado ha sido probado con los tipos de datos REAL y DOBLE, habiéndose obtenido los resultados siguientes en las cuatro ultimas iteraciones:

TIPO REAL

Iteración 37: x = 7.27595761418343E−12

Iteración 38: x = 3.36797880709171E−12

Iteración 39: x = 1.81898940354586E−12

Iteración 40: x = 9.09494701772928E−13

El numero mas pequeño que aporta información a uno es:

9.09494701772928E−13

TIPO DOBLE

Iteración 49: x = 1.77635683940025E−15

Iteración 50: x = 8.88178419700125E−16

Iteración 51: x = 4.40892095006300E−16

Iteración 52: x = 2.22044604925031E−16

El numero mas pequeño que aporta información a uno es:

2.22044604925031E−16

El funcionamiento del programa es el siguiente:

Asignamos a una variable el valor 1, y los vamos dividiendo por 2 en cada interacción, lo que supones un desplazamiento a la derecha de la mantisa, por lo que los valores obtenidos realmente no son los valores mínimos que aportan información a uno, sino los que podemos representan en los formatos REAL y DOBLE.

LISTADO DEL PROGRAMA

```
Program ejer1;

uses

wincrt;
```

```
var

x,y:real;

j,k:double;

i:integer;

−−>[Author:MIS0]begin

x:=1;

y:=0;

i:=0;

while y<>1 do

begin

i:=i+1;

x:=x/2;

y:=1+x;

writeln('Numero de iteración: ',i,' El valor de la x es: ',x,' El valor de la y es: ',y);

end;

writeln;

writeln('El minimo numero que aporta información a 1 para tipo real es: ',x*2,' en la iteracion: ',i−1);

readln;

j:=1;

k:=0;

i:=0;

while k<>1 do

begin

i:=i+1;

j:=j/2;

k:=1+j;
```

```
writeln('Numero de iteración: ',i,' El valor de la j es: ',j,' El valor de la k es: ',k);

end;

writeln;

writeln('El minimo numero que aporta información a 1 para tipo double es: ',j*2,' en la iteracion: ',i−1);

end.
```

MODIFICACIÓN

Si modificamos el programa para que en cada iteracion el numero almacenado en x se divida por 10 en vez de por 2, los resultados obtenidos son los siguientes:

TIPO REAL: El numero mas pequeño que aporta información a 1 es 9.99999999999273E−14 y lo hemos obtenido en la iteracion 13.

TIPO DOUBLE: El numero mas pequeño que aporta información a uno es 1.00000000000000E−16, en la iteracion 16.

Como podemos observar el numero de iteraciones obtenidas al realizar la modificación del programa es mucho menor que en el programa original, sin embargo, la información obtenida en esta modificación es menos fiable ya que no son desplazamientos exactos de la mantisa, cosa que si ocurría en el programa original.

LISTADO DEL PROGRAMA

```
Program ejer1;

uses

wincrt;

var

x,y:real;

j,k:double;

i:integer;

begin

x:=1;

y:=0;

i:=0;

while y<>1 do
```

```
begin
i:=i+1;
x:=x/10;
y:=1+x;
writeln('Numero de iteración: ',i,' El valor de la x es: ',x,' El valor de la y es: ',y);
end;
readln;
writeln;
writeln('El minimo numero que aporta información a 1 para tipo real es: ',x*10,' en la iteracion: ',i−1);
j:=1;
k:=0;
i:=0;
while k<>1 do
begin
i:=i+1;
j:=j/10;
k:=1+j;
writeln('Numero de iteración: ',i,' El valor de la j es: ',j,' El valor de la k es: ',k);
end;
writeln;
writeln('El minimo numero que aporta información a 1 para tipo double es: ',j*10,' en la iteracion: ',i−1);
end.
```

EJERCICIO 2

El objetivo del presente programa es obtener el minimo numero que se puede representar tanto en formato REAL, como en formato DOUBLE.

Una vez ejecutado el programa los resultados obtenido han sido los siguientes.

TIPO REAL:

Iteracion 126: x=1.17549435082229E−38

Iteracion 127: x=5.87747175411144E−38

Iteracion 128: x=2.93873587705572E−39

El minimo numero que se puede representar en este formato es:

2.93873587705572E−39

TIPO DOUBLE:

Iteracion 1072: x=1.97626258336499E−323

Iteracion 1073: x=9.88131291682493E−324

Iteracion 1074: x=4.94065645841247E−324

El minimo numero que se puede representar en este formato es:

4.94065645841247E−324

LISTADO DEL PROGRAMA

```
Program ejer2;

uses

wincrt;

var

x,y:real;

j,k:double;

i:integer;

begin

x:=1;

y:=0;

i:=0;

repeat

i:=i+1;

y:=x;
```

```
x:=x/2;

writeln('Iteracion numero ',i,'x:= ',x);

until x=0;

writeln('El minimo numero que se puede representar en formato real es: ',y);

readln;

j:=1;

i:=0;

k:=0;

repeat

i:=i+1;

k:=j;

j:=j/2;

writeln('Iteracion numero ',i,'x:= ',j);

until j=0;

writeln;

writeln('El minimo numero que se puede representar en formato double es:',k);

end.
```

EJERCICIO 3

En este ejercicio hemos realizado un programa para resolver un sistema de ecuaciones utilizando la regla de Cramer, y trabajando en todo momento con el tipo REAL.

El sistema de ecuaciones a resolver es el siguiente:

10000000x + y = 10000001

10000001x + y = 10000002

Una vez ejecutado el programa los resultados obtenidos han sido los siguientes

x = 1;

y = 0;

A continuación hemos ejecutado el mismo programa pero realizando una pequeña variación en el sistema de

ecuaciones a resolver:

1000000x + y = 1000001

1000001x + y = 1000002

Los resultados obtenidos han sido los que mostramos a continuación:

x= 1;

y= 1;

Dado que no hemos obtenido los mismos resultados en las dos ejecuciones, lo cual no es correcto, hemos procedido ha realizar una traza para ver en que momento de la ejecución del programa se produce el error.

La traza para el primer sistema es la siguiente:

mult1 = 10000001

mult2 = 10000002

mult3 = 1.0000002E+14

mult4 = 1.0000002E+14

res1 = −1

res2 = 0

div1 = 1

div2 = 0

La traza para el segundo sistema es la siguiente:

mult1 = 1000001

mult2 = 1000002

mult3 = 1.000002E+12

mult4 = 1.000002E+12

res1 = −1

res2 = −1

div1 = 1

div2 = 1

Los resultados de las variable x e y son los obtenidos en div1 y div2 que en el primer sistema nos son

correctos ya que el valor obtenido en res2 que corresponde a la diferencia entre mult3 y mult4 es erroneo ya que dichos valores exceden del rango del tipo de datos REAL, por lo que el resultado correcto es el que obtenemos en la resolución del segundo sistema de ecuaciones.

LISTADO DEL PROGRAMA

```
Program cramer;

uses

wincrt;

const

t=2;

type

matriz=array [1..t,1..t] of real;

vector=array [1..t] of real;

var

m:matriz;

v1,v2:vector;

det,mult1,mult2,mult3,mult4,res1,res2,div1,div2:real;

i,j:integer;

function determinante (m:matriz):real;

var

i,j:integer;

begin

determinante:=m[1,1]*m[2,2]−m[1,2]*m[2,1];

end;

begin

for i:=1 to t do

for j:=1 to t do

begin
```

```
write('Introduce el coeficiente ',i,',',j,': ');

readln(m[i,j]);

end;

for i:=1 to t do

begin

write('Introduce el termino independiente ',i,': ');

readln(v1[i]);

end;

det:=determinante(m);

mult1:=v1[1]*m[2,2];

mult2:=v1[2]*m[1,2];

res1:=mult1−mult2;

div1:=res1/det;

v2[1]:=div1;

mult3:=v1[2]*m[1,1];

mult4:=v1[1]*m[2,1];

res2:=mult3−mult4;

div2:=res2/det;

v2[2]:=div2;

writeln('El valor de x es: ',v2[1]);

writeln('El valor de y es: ',v2[2]);

end.
```

EJERCICIO 4

En este ejercicio se pretende realizar el producto de dos matrices de enteros, utilizando para ello distintos tamaños de las mismas, para poder evaluar el rendimiento del algoritmo tanto a priorio como a posteriori.

Para realizar el análisis a priori hemos utilizado como unidad de medida el FLOP, que es el tiempo que tarde en ejecutarse una de las operaciones elementales entre números en coma flotante.

Para realizar el análisis a posteriori hemos usado la función GETTIME que nos permite capturar los tiempos antes y después de la ejecución de los métodos a evaluar.

ANÁLISIS A PRIORI

Hemos observado que el procedimiento utilizado para la obtención del producto de dos matrices utiliza tres bucles FOR anidados, que contienen las operaciones realizadas en coma flotante y que nos van a determinar el numero de FLOPs , además , la talla de los mismos depende del tamaño de las matrices.

Siendo N el tamaño de la matriz y teniendo en cuenta que en cada iteracion hemos realizado dos operaciones en coma flotante, el numero de FLOPs es:

N*N*N*2

Por lo tanto el coste del algoritmo es:

O(n3)

El numero de flops para cada tamaño de las matrices se puede observar en la siguiente tabla:

TAMAÑO	FLOPS
8*8	1024
16*16	8192
24*24	27648
32*32	65536
40*40	128000
48*48	221184
56*56	351232
64*64	524288

A continuacion se puede observar la grafica del coste del algoritmo

para cada tamaño de las matrices, medido en flops: