



# INFORMATICA I

Serie del número de Euler en "C"

Ing. Juan Carlos Cuttitta

*Universidad Tecnológica Nacional  
Facultad Regional Buenos Aires  
Departamento de Ingeniería Electrónica*

4 de mayo de 2020



# Enunciado del problema

El valor aproximado del número de Euler ( $e$ ) se puede obtener con la siguiente serie.

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

$$e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \dots$$

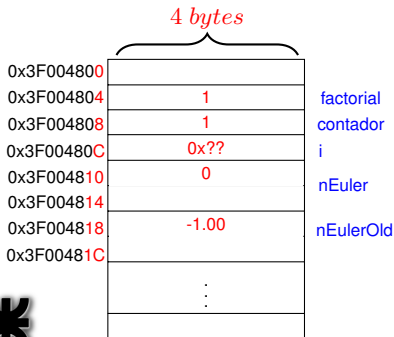
Escribir un programa que calcule el valor aproximado de  $e$  mediante un ciclo repetitivo que termine cuando la diferencia entre dos aproximaciones sucesivas difiera en menos de  $10^{-9}$



# Declaración y disposición en memoria

## Arquitectura X86-32 bits

*Disposición de las variables en la memoria*



## Código del programa fuente

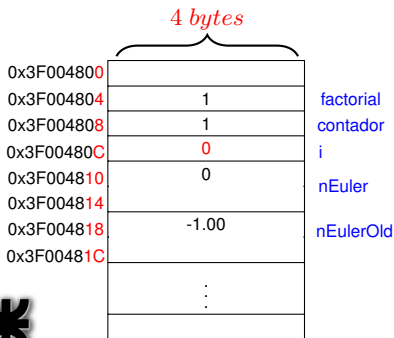
```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int factorial = 1, contador = 1, i;
8     double nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Declaración y disposición en memoria

## Arquitectura X86-32 bits

*Declara la variable  $i$*



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*Pregunta condición del **for***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	0	i
0x3F004810	0	nEuler
0x3F004814		
0x3F004818	-1.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int factorial = 1, contador = 1, i;
8     double nEuler=0, nEulerOld=-1;
9
10    for(i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while(contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

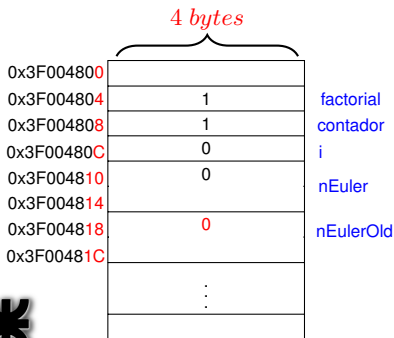
*Diagrama: Una línea roja curva apunta desde el comentario  $10^{-9}$  hacia el valor `LIMITE` en la línea 4 del código.*



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Para conservar el *e* anterior



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* no es  $\leq$  a *i*  
no entra dentro del *while*

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	0	i
0x3F004810	0	nEuler
0x3F004814		
0x3F004818	0	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

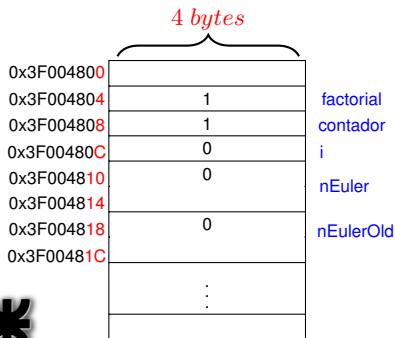
```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Se debe *castear* la variable para que ambas sean *double*



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double) factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*almacena el resultado en la variable **nEuler***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	0	i
0x3F004810	1.00	nEuler
0x3F004814		
0x3F004818	0	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double) factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

$0 + \frac{1}{1,00} = 1,00$



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*inicializo nuevamente las variables **factorial** y **contador***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	0	i
0x3F004810	1.00	nEuler
0x3F004814		
0x3F004818	0	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

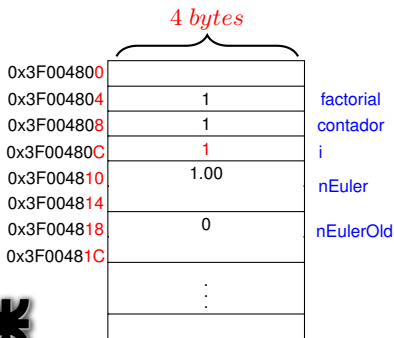
```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable  $i$*



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*Pregunta condición del **for***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	0	i
0x3F004810	1.00	nEuler
0x3F004814		
0x3F004818	0	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

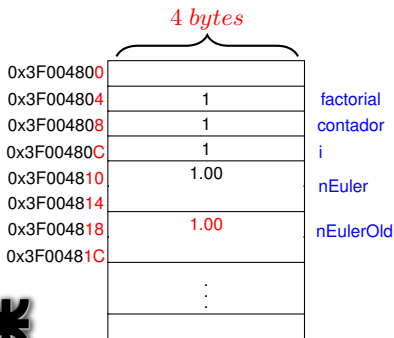
```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE  $\overbrace{\text{pow}(10,-9)}^{10^{-9}}$ 
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; nEuler-nEulerOld >= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Para conservar el *e* anterior



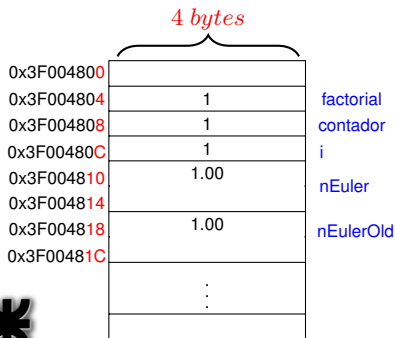
## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* es  $\leq$  a *i*  
entra dentro del *while*



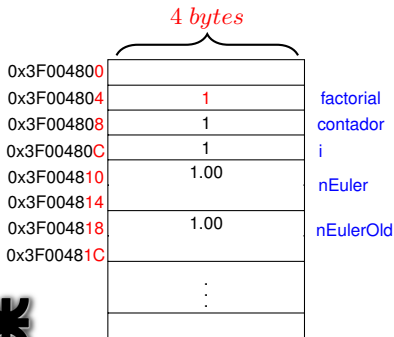
## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

almacena el resultado en la variable *factorial*



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for(i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while(contador <= i) 1 * 1 = 1
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable **contador***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	2	contador
0x3F00480C	1	i
0x3F004810	1.00	nEuler
0x3F004814		
0x3F004818	1.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```





# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* no es  $\leq$  a *i*  
sale del loop *while*

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	2	contador
0x3F00480C	1	i
0x3F004810	1.00	nEuler
0x3F004814		
0x3F004818	1.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

almacena el resultado en la variable *nEuler*

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	2	contador
0x3F00480C	1	i
0x3F004810	2.00	nEuler
0x3F004814		
0x3F004818	1.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for(i=0;(nEuler-nEulerOld)>= LIMITE;i++)
11    {
12        nEulerOld=nEuler;
13        while(contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double) factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

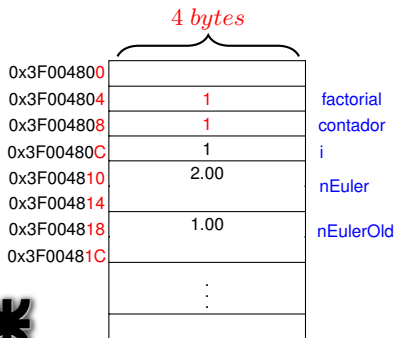
$1,00 + \frac{1}{1,00} = 2,00$



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*inicializo nuevamente las variables **factorial** y **contador***



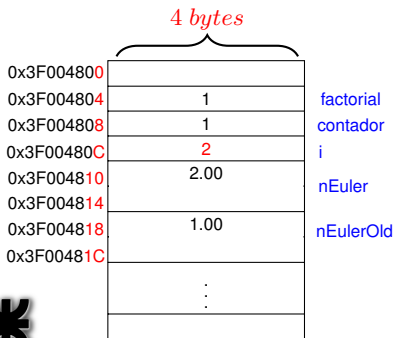
## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable  $i$*



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*Pregunta condición del **for***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	2	i
0x3F004810	2.00	nEuler
0x3F004814		
0x3F004818	1.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

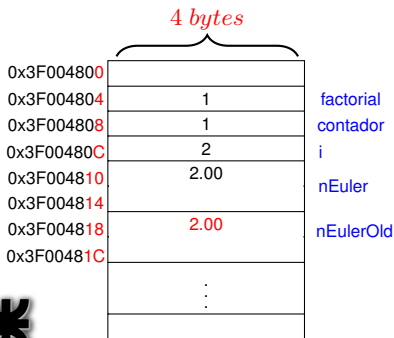
```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE  $\overbrace{\text{pow}(10,-9)}^{10^{-9}}$ 
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; nEuler-nEulerOld >= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Para conservar el *e* anterior



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* es  $\leq$  a *i*  
entra dentro del *while*

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	2	i
0x3F004810	2.00	nEuler
0x3F004814		
0x3F004818	2.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*almacena el resultado en la variable **factorial***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	2	i
0x3F004810	2.00	nEuler
0x3F004814		
0x3F004818	2.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for(i=0;(nEuler-nEulerOld)>= LIMITE;i++)
11    {
12        nEulerOld=nEuler;
13        while(contador <= i) 1 · 1 = 1
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```





# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable **contador***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	2	contador
0x3F00480C	2	i
0x3F004810	2.00	nEuler
0x3F004814		
0x3F004818	2.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %.10f \n", nEuler);
24    printf("e(lib math.h)es %.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* es  $\leq$  a *i*  
entra dentro del *while*

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	2	contador
0x3F00480C	2	i
0x3F004810	2.00	nEuler
0x3F004814		
0x3F004818	2.00	nEulerOld
0x3F00481C		
	⋮	

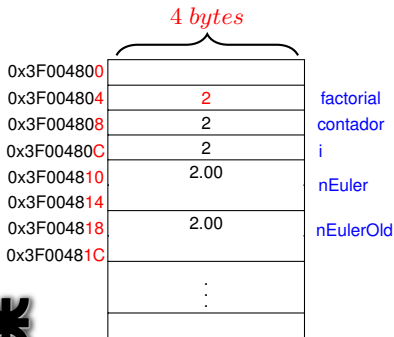
## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

almacena el resultado en la variable *factorial*



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i) 1 · 2 = 2
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable **contador***

4 bytes

0x3F004800		
0x3F004804	2	factorial
0x3F004808	3	contador
0x3F00480C	2	i
0x3F004810	2.00	nEuler
0x3F004814		
0x3F004818	2.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* no es  $\leq$  a *i*  
sale del loop *while*

4 bytes

0x3F004800		
0x3F004804	2	factorial
0x3F004808	3	contador
0x3F00480C	2	i
0x3F004810	2.00	nEuler
0x3F004814		
0x3F004818	2.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*almacena el resultado en la variable **nEuler***

4 bytes

0x3F004800		
0x3F004804	2	factorial
0x3F004808	3	contador
0x3F00480C	2	i
0x3F004810	2.50	nEuler
0x3F004814		
0x3F004818	2.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for(i=0;(nEuler-nEulerOld)>= LIMITE;i++)
11    {
12        nEulerOld=nEuler;
13        while(contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double) factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

*2,00 +  $\frac{1}{2,00} = 2,50$*

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*inicializo nuevamente las variables **factorial** y **contador***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	2	i
0x3F004810	2.50	nEuler
0x3F004814		
0x3F004818	2.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for(i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while(contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable  $i$*

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	3	$i$
0x3F004810	2.50	nEuler
0x3F004814		
0x3F004818	2.00	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %.10f \n", nEuler);
24    printf("e(lib math.h)es %.10f\n", M_E);
25    return 0;
26 }
```





# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*Pregunta condición del **for***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	3	i
0x3F004810	2.50	nEuler
0x3F004814		
0x3F004818	2.00	nEulerOld
0x3F00481C		
	⋮	

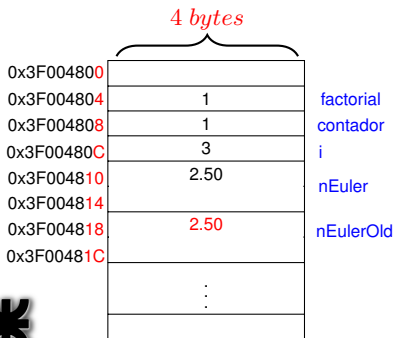
## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE  $\overbrace{\text{pow}(10,-9)}^{10^{-9}}$ 
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; nEuler-nEulerOld >= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Para conservar el *e* anterior



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* es  $\leq$  a *i*  
entra dentro del *while*

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	3	i
0x3F004810	2.50	nEuler
0x3F004814		
0x3F004818	2.50	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

almacena el resultado en la variable *factorial*

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	3	i
0x3F004810	2.50	nEuler
0x3F004814		
0x3F004818	2.50	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i) 1 · 1 = 1
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable **contador***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	2	contador
0x3F00480C	3	i
0x3F004810	2.50	nEuler
0x3F004814		
0x3F004818	2.50	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %.10f \n", nEuler);
24    printf("e(lib math.h)es %.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* es  $\leq$  a *i*  
entra dentro del *while*

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	2	contador
0x3F00480C	3	i
0x3F004810	2.50	nEuler
0x3F004814		
0x3F004818	2.50	nEulerOld
0x3F00481C		
	⋮	

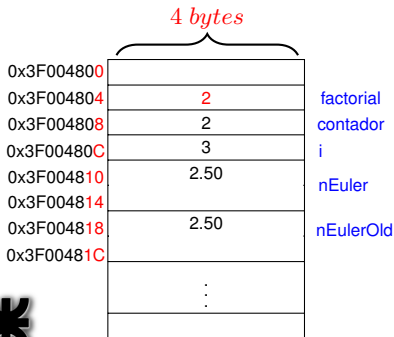
## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

almacena el resultado en la variable *factorial*



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for(i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while(contador <= i) 1 · 2 = 2
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable **contador***

4 bytes

0x3F004800		
0x3F004804	2	factorial
0x3F004808	3	contador
0x3F00480C	3	i
0x3F004810	2.50	nEuler
0x3F004814		
0x3F004818	2.50	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

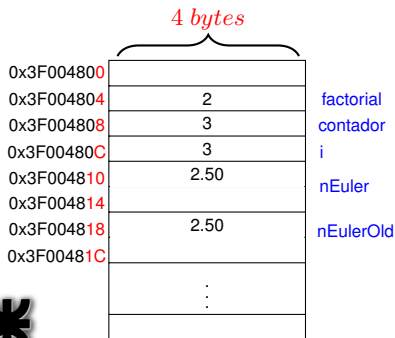




# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* es  $\leq$  a *i*  
entra dentro del *while*



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

almacena el resultado en la variable *factorial*

4 bytes

0x3F004800		
0x3F004804	6	factorial
0x3F004808	3	contador
0x3F00480C	3	i
0x3F004810	2.50	nEuler
0x3F004814		
0x3F004818	2.50	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i) 2 * 3 = 6
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double) factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable **contador***

4 bytes

0x3F004800		
0x3F004804	6	factorial
0x3F004808	4	contador
0x3F00480C	3	i
0x3F004810	2.50	nEuler
0x3F004814		
0x3F004818	2.50	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* no es  $\leq$  a *i*  
sale del loop *while*

4 bytes

0x3F004800		
0x3F004804	6	factorial
0x3F004808	4	contador
0x3F00480C	3	i
0x3F004810	2.50	nEuler
0x3F004814		
0x3F004818	2.50	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

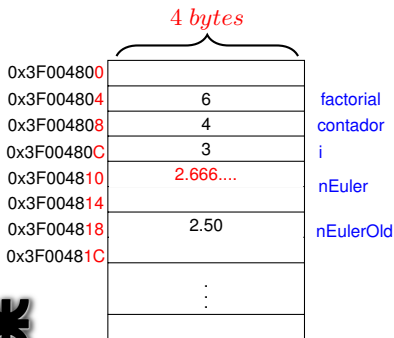
```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

almacena el resultado en la variable *nEuler*



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for(i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while(contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double) factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

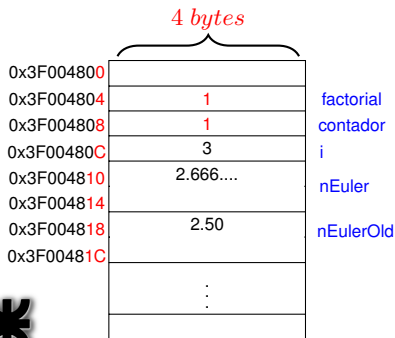
$2,50 + \frac{1}{6,00} = 2,6\hat{6}$



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*inicializo nuevamente las variables **factorial** y **contador***



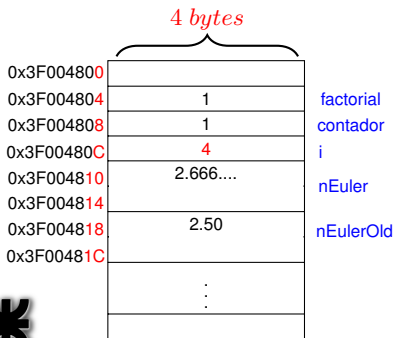
## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable  $i$*



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*Pregunta condición del for*

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	4	i
0x3F004810	2.666....	nEuler
0x3F004814	2.50	nEulerOld
0x3F004818		
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE  $\overbrace{\text{pow}(10,-9)}^{10^{-9}}$ 
5 int main(void)
6 {
7     int factorial = 1, contador = 1, i;
8     double nEuler=0, nEulerOld=-1;
9
10    for(i=0; nEuler-nEulerOld >= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while(contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

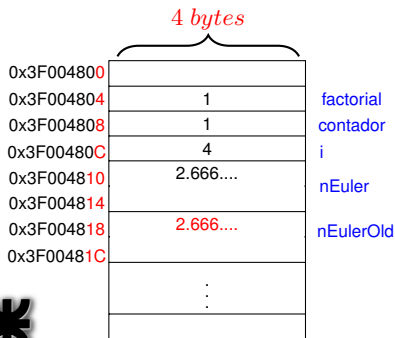




# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Para conservar el *e* anterior



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %.10f \n", nEuler);
24    printf("e(lib math.h)es %.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* es  $\leq$  a *i*  
entra dentro del *while*

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	4	i
0x3F004810	2.666....	nEuler
0x3F004814		
0x3F004818	2.666....	nEulerOld
0x3F00481C		
	⋮	

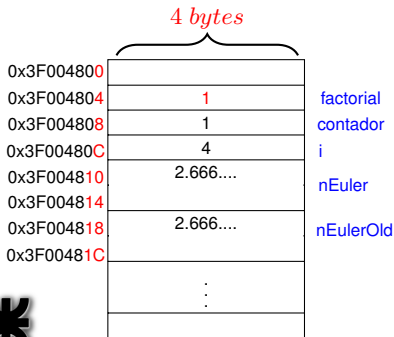
## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

almacena el resultado en la variable *factorial*



## Código del programa fuente

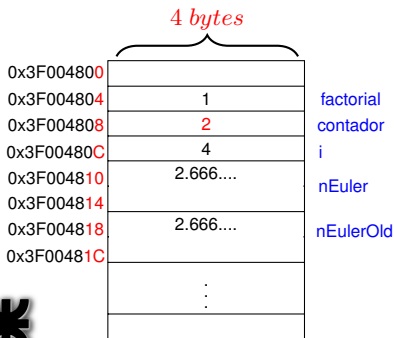
```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i) 1 · 1 = 1
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable **contador***



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %.10f \n", nEuler);
24    printf("e(lib math.h)es %.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* es  $\leq$  a *i*  
entra dentro del *while*

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	2	contador
0x3F00480C	4	i
0x3F004810	2.666....	nEuler
0x3F004814		
0x3F004818	2.666....	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

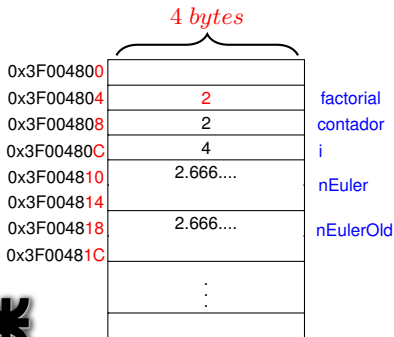
```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

almacena el resultado en la variable *factorial*



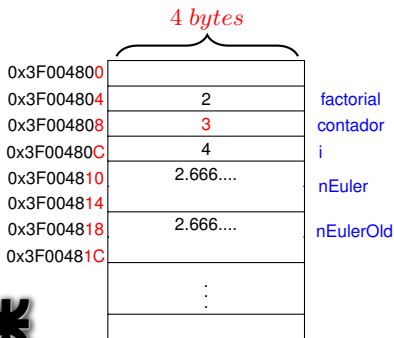
## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i) 1 · 2 = 2
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable **contador***



## Código del programa fuente

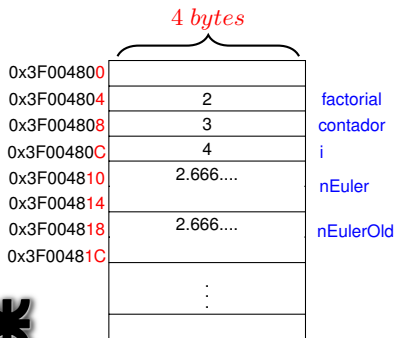
```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %.10f \n", nEuler);
24    printf("e(lib math.h)es %.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* es  $\leq$  a *i*  
entra dentro del *while*



## Código del programa fuente

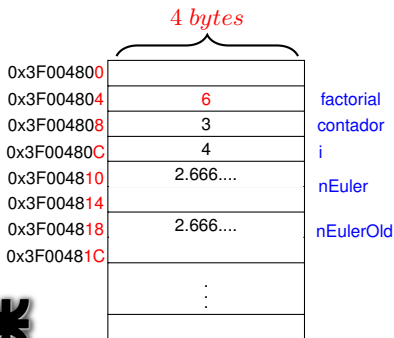
```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

almacena el resultado en la variable *factorial*



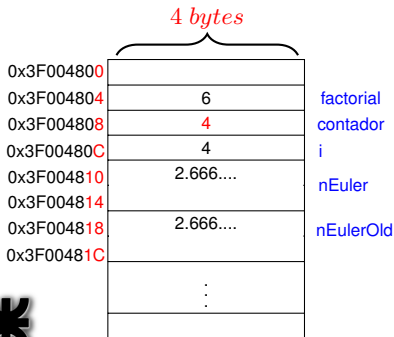
## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i) 2 * 3 = 6
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable **contador***



## Código del programa fuente

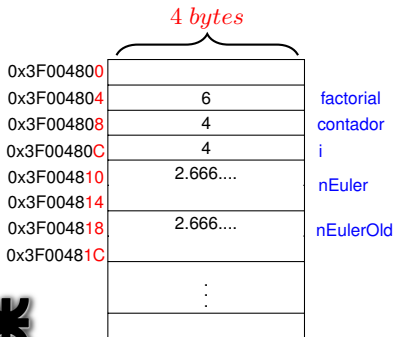
```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* es  $\leq$  a *i*  
entra dentro del *while*



## Código del programa fuente

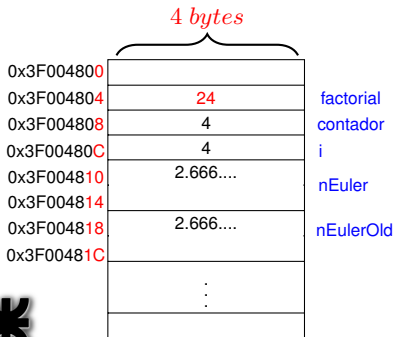
```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

almacena el resultado en la variable *factorial*



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double) factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

$6 \cdot 4 = 24$



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable **contador***

	4 bytes	
0x3F004800		
0x3F004804	24	factorial
0x3F004808	5	contador
0x3F00480C	4	i
0x3F004810	2.666....	nEuler
0x3F004814		
0x3F004818	2.666....	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %.10f \n", nEuler);
24    printf("e(lib math.h)es %.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

Como *contador* no es  $\leq$  a *i*  
sale del loop *while*

4 bytes

0x3F004800		
0x3F004804	24	factorial
0x3F004808	5	contador
0x3F00480C	4	i
0x3F004810	2.666....	nEuler
0x3F004814		
0x3F004818	2.666....	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*almacena el resultado en la variable **nEuler***

4 bytes

0x3F004800		
0x3F004804	24	factorial
0x3F004808	5	contador
0x3F00480C	4	i
0x3F004810	2.7083....	nEuler
0x3F004814		
0x3F004818	2.666....	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/((double) factorial));
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

$2,66 + \frac{1}{24,00} = 2,708\overline{3}$



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*inicializo nuevamente las variables **factorial** y **contador***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	4	i
0x3F004810	2.7083....	nEuler
0x3F004814		
0x3F004818	2.666....	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/((double)factorial));
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```

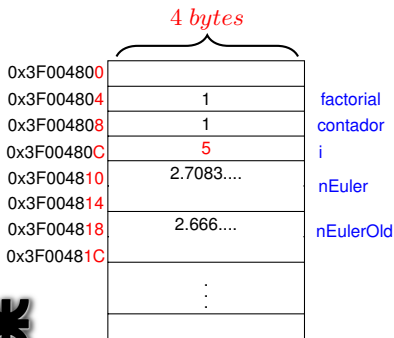
$2,66 + \frac{1}{24,00} = 2,7083$



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*incremento la variable  $i$*



## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %.10f \n", nEuler);
24    printf("e(lib math.h)es %.10f\n", M_E);
25    return 0;
26 }
```

# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*Pregunta condición del **for***

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	5	i
0x3F004810	2.7083....	nEuler
0x3F004814		
0x3F004818	2.666....	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE  $\overbrace{\text{pow}(10,-9)}^{10^{-9}}$ 
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; nEuler-nEulerOld >= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }
```



Si seguimos observando paso a paso como se va ejecutando el programa y como van cambiando los valores de las variables de memoria, llegaremos al punto donde en el ciclo del **for** no se va a cumplir la condición

$$(nEuler - nEulerOld) \geq LIMITE$$

y por lo tanto los valores de memoria tendrán los siguientes valores



# Recorriendo el programa **paso a paso**

## Arquitectura X86-32 bits

*No cumple condición del for*  
( $nEuler - nEulerOld$ ) es **0,0000000005**

4 bytes

0x3F004800		
0x3F004804	1	factorial
0x3F004808	1	contador
0x3F00480C	13	i
0x3F004810	2.7182818288	nEuler
0x3F004814		
0x3F004818	2.7182818283	nEulerOld
0x3F00481C		
	⋮	

## Código del programa fuente

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE  $\overbrace{\text{pow}(10,-9)}^{10^{-9}}$ 
5 int main(void)
6 {
7     int factorial = 1, contador = 1, i;
8     double nEuler=0, nEulerOld=-1;
9
10    for(i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while(contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %.10f \n", nEuler);
24    printf("e(lib math.h)es %.10f\n", M_E);
25    return 0;
26 }
```



## Arquitectura X86-32 bits

*e* es 2,7182818288*e* (lib math.h) es 2,7182818285

## Código del programa fuente

```

1 #include <stdio.h>
2 #include <math.h>
3
4 #define LIMITE pow(10,-9)
5 int main(void)
6 {
7     int     factorial = 1, contador = 1, i;
8     double  nEuler=0, nEulerOld=-1;
9
10    for (i=0; (nEuler-nEulerOld)>= LIMITE; i++)
11    {
12        nEulerOld=nEuler;
13        while (contador <= i)
14        {
15            factorial = factorial * contador;
16            contador++;
17        }
18        nEuler = nEuler + (1/(double)factorial);
19
20        factorial = 1;
21        contador = 1;
22    }
23    printf("e es %0.10f \n", nEuler);
24    printf("e(lib math.h)es %0.10f\n", M_E);
25    return 0;
26 }

```