Tarea 1 **Calculo Computacional**

Victor Tortolero CI:24.569.609

Respuesta 1

Tenemos que $\frac{A+3}{13}$, como $A = 9$, tendríamos $\frac{9+3}{13} = \frac{12}{13}$. Ahora procedemos a convertir a binario.

$\frac{12}{13} \times 2 = \frac{24}{13}$, $b_0 = 1$
$\frac{11}{13} \times 2 = \frac{22}{13}$, $b_1 = 1$
$\frac{9}{13} \times 2 = \frac{18}{13}$, $b_2 = 1$
$\frac{5}{13} \times 2 = \frac{10}{13}$, $b_3 = 0$
$\frac{10}{13} \times 2 = \frac{20}{13}$, $b_4 = 1$
$\frac{7}{13} \times 2 = \frac{14}{13}$, $b_5 = 1$
$\frac{1}{13} \times 2 = \frac{2}{13}$, $b_6 = 0$
$\frac{2}{13} \times 2 = \frac{4}{13}$, $b_7 = 0$
$\frac{4}{13} \times 2 = \frac{8}{13}$, $b_8 = 0$
$\frac{8}{13} \times 2 = \frac{16}{13}$, $b_9 = 1$
$\frac{3}{13} \times 2 = \frac{6}{13}$, $b_{10} = 0$
$\frac{6}{13} \times 2 = \frac{12}{13}$, $b_{11} = 0$

Por lo tanto tenemos que:

$$0{,}1110110001001\overline{110110001001}\mathbf{1}\ldots$$

Observemos que el numero que vendria luego del bit 24 seria un 1. Entonces a la hora de redondear se suma 1. Por lo tanto, tenemos que $Fl(\frac{12}{13})_{Truncado} = 0{,}11101100010011011000100$, y que $Fl(\frac{12}{13})_{Redondeado} = 0{,}11101100010011011000101$.

Por Truncamiento tenemos que:

$$\begin{aligned} E_A = |x - Fl(x)_{Truncado}| &= 0,\underbrace{000\ldots000}_{\text{24 Ceros}}111011000100\ldots \\ &= 0,\underbrace{111011000100111011000100}_{\text{Esto es } \frac{12}{13}}\cdots \times 2^{-24} \\ &= \frac{12}{13} \times 2^{-24} \approx 5,50196 \times 10^{-8} \\ E_R = \frac{E_A}{|x|} &= \frac{\frac{12}{13} \times 2^{-24}}{\frac{12}{13}} \\ &= 2^{-24} \approx 5,96046 \times 10^{-8} \end{aligned}$$

Por Redondeo tenemos que:

$$\begin{aligned} E_A = |x - Fl(x)_{Redondeado}| &= |x - (Fl(x)_{Truncado} + 1 \times 2^{-24})| \\ &= |x - Fl(x)_{Truncado} - 1 \times 2^{-24}| \\ &= |\frac{12}{13} \times 2^{-24} - 1 \times 2^{-24}| \\ &= |\frac{12}{13} - 1| \times 2^{-24} \\ &= \frac{1}{13} \times 2^{-24} \approx 4,584 \times 10^{-9} \end{aligned}$$

$$\begin{aligned} E_R = \frac{E_A}{|x|} &= \frac{\frac{1}{13} \times 2^{-24}}{\frac{12}{13}} \\ &= \frac{1}{12} \times 2^{-24} \approx 4,9670 \times 10^{-9} \end{aligned}$$

Respuesta 2

Tenemos $245696,09_{10}$, procedemos a convertirlo a binario:

- **Parte Entera**:

$\frac{245696}{2} = 122848,\ b_{17} = 0$
$\frac{122848}{2} = 61424,\ b_{16} = 0$
$\frac{61424}{2} = 30712,\ b_{15} = 0$
$\frac{30712}{2} = 15356,\ b_{14} = 0$
$\frac{15356}{2} = 7678,\ b_{13} = 0$
$\frac{7678}{2} = 3839,\ b_{12} = 0$
$\frac{3839}{2} = 1919,\ b_{11} = 1$
$\frac{1919}{2} = 959,\ b_{10} = 1$
$\frac{959}{2} = 479,\ b_{09} = 1$
$\frac{479}{2} = 239,\ b_{08} = 1$

$\frac{239}{2} = 119,\ b_{07} = 1$
$\frac{119}{2} = 59,\ b_{06} = 1$
$\frac{59}{2} = 29,\ b_{05} = 1$
$\frac{29}{2} = 14,\ b_{04} = 1$
$\frac{14}{2} = 7,\ b_{03} = 0$
$\frac{7}{2} = 7,\ b_{02} = 1$
$\frac{3}{2} = 1,\ b_{01} = 1$
$\frac{1}{2} = 0,\ b_{00} = 1$

Por lo tanto tenemos que $245696_{10} = 111011111111000000_2$.

- **Parte Decimal**:

$\frac{9}{100} \times 2 = \frac{18}{100},\ b_0 = 0$
$\frac{18}{100} \times 2 = \frac{36}{100},\ b_1 = 0$
$\frac{36}{100} \times 2 = \frac{27}{100},\ b_2 = 0$
$\frac{72}{100} \times 2 = \frac{144}{100},\ b_3 = 1$
$\frac{44}{100} \times 2 = \frac{88}{100},\ b_4 = 0$

$\frac{88}{100} \times 2 = \frac{176}{100},\ b_5 = 1$
$\frac{76}{100} \times 2 = \frac{152}{100},\ b_6 = 1$
$\frac{52}{100} \times 2 = \frac{104}{100},\ b_7 = 1$

Por lo tanto tenemos que $0,09_{10} = 00010111$.

Entonces se tiene que $245696,09_{10} \approx \overbrace{111011111111000000,000101}^{24bits}11_2$.

Si usamos **redondeo:**
$\boldsymbol{Fl(245696,09)_{Redondeado} = 0,11101111111100000000110 \times 2^{18}}$

Si representamos este numero de vuelta en **decimal:**
$\mathbf{111011111111000000,000110_2 = 245696,09375_{10}}$.

Error absoluto y relativo:

$$\begin{aligned} E_A = |x - Fl(x)_{Redondeado}| &= |245696,09 - 245696,09375| \\ &= -3{,}75 \times 10^{-3} \\ E_R = \frac{E_A}{|x|} &= \frac{-3,75 \times 10^{-3}}{245696,09} \approx -1{,}526275815 \times 10^{-8} \end{aligned}$$

Respuesta 3

Después de correr el programa, se obtuvieron los siguientes datos

- **Para simple precisión**: $\epsilon = 0{,}00000011920928955078125000000000000$

Iteracion	t	ϵ
1	1.500000000000000000000000	0.500000000000000000000000000000000000
2	1.250000000000000000000000	0.250000000000000000000000000000000000
3	1.125000000000000000000000	0.125000000000000000000000000000000000
4	1.062500000000000000000000	0.062500000000000000000000000000000000
5	1.031250000000000000000000	0.031250000000000000000000000000000000
6	1.015625000000000000000000	0.015625000000000000000000000000000000
7	1.007812500000000000000000	0.007812500000000000000000000000000000
8	1.003906250000000000000000	0.003906250000000000000000000000000000
9	1.001953125000000000000000	0.001953125000000000000000000000000000
10	1.000976562500000000000000	0.000976562500000000000000000000000000
11	1.000488281250000000000000	0.000488281250000000000000000000000000
12	1.000244140625000000000000	0.000244140625000000000000000000000000
13	1.000122070312500000000000	0.000122070312500000000000000000000000
14	1.000061035156250000000000	0.000061035156250000000000000000000000
15	1.000030517578125000000000	0.000030517578125000000000000000000000
16	1.000015258789062500000000	0.000015258789062500000000000000000000

17	1.000007629394531200000000	0.000007629394531250000000000000000000
18	1.000003814697265600000000	0.000003814697265625000000000000000000
19	1.000001907348632800000000	0.000001907348632812500000000000000000
20	1.000000953674316400000000	0.000000953674316406250000000000000000
21	1.000000476837158200000000	0.000000476837158203125000000000000000
22	1.000000238418579100000000	0.000000238418579101562500000000000000
23	1.000000119209289600000000	0.000000119209289550781250000000000000
24	1.000000000000000000000000	0.000000059604644775390625000000000000

- **Para doble precisión**: $\epsilon = 0{,}00000000000000022204460492503131 00$

Iteracion	t	ϵ
1	1.500000000000000000000000	0.500000000000000000000000000000000000
2	1.250000000000000000000000	0.250000000000000000000000000000000000
3	1.125000000000000000000000	0.125000000000000000000000000000000000
4	1.062500000000000000000000	0.062500000000000000000000000000000000
5	1.031250000000000000000000	0.031250000000000000000000000000000000
6	1.015625000000000000000000	0.015625000000000000000000000000000000
7	1.007812500000000000000000	0.007812500000000000000000000000000000
8	1.003906250000000000000000	0.003906250000000000000000000000000000
9	1.001953125000000000000000	0.001953125000000000000000000000000000
10	1.000976562500000000000000	0.000976562500000000000000000000000000
11	1.000488281250000000000000	0.000488281250000000000000000000000000
12	1.000244140625000000000000	0.000244140625000000000000000000000000
13	1.000122070312500000000000	0.000122070312500000000000000000000000
14	1.000061035156250000000000	0.000061035156250000000000000000000000
15	1.000030517578125000000000	0.000030517578125000000000000000000000
16	1.000015258789062500000000	0.000015258789062500000000000000000000
17	1.000007629394531200000000	0.000007629394531250000000000000000000
18	1.000003814697265600000000	0.000003814697265625000000000000000000
19	1.000001907348632800000000	0.000001907348632812500000000000000000
20	1.000000953674316400000000	0.000000953674316406250000000000000000
21	1.000000476837158200000000	0.000000476837158203125000000000000000
22	1.000000238418579100000000	0.000000238418579101562500000000000000
23	1.000000119209289600000000	0.000000119209289550781250000000000000
24	1.000000059604644800000000	0.000000059604644775390625000000000000
25	1.000000029802322400000000	0.000000029802322387695313000000000000
26	1.000000014901161200000000	0.000000014901161193847656000000000000
27	1.000000007450580600000000	0.000000007450580596923828100000000000

28	1.000000003725290300000000	0.000000003725290298461914100000000
29	1.000000001862645100000000	0.000000001862645149230957000000000
30	1.000000000931322600000000	0.000000000931322574615478520000000
31	1.000000000465661300000000	0.000000000465661287307739260000000
32	1.000000000232830600000000	0.000000000232830643653869630000000
33	1.000000000116415300000000	0.000000000116415321826934810000000
34	1.000000000058207700000000	0.000000000058207660913467407000000
35	1.000000000029103800000000	0.000000000029103830456733704000000
36	1.000000000014551900000000	0.000000000014551915228366852000000
37	1.000000000007276000000000	0.000000000007275957614183425900000
38	1.000000000003638000000000	0.000000000003637978807091713000000
39	1.000000000001819000000000	0.000000000001818989403545856500000
40	1.000000000000909500000000	0.000000000000909494701772928240000
41	1.000000000000454700000000	0.000000000000454747350886464120000
42	1.000000000000227400000000	0.000000000000227373675443232060000
43	1.000000000000113700000000	0.000000000000113686837721616030000
44	1.000000000000056800000000	0.000000000000056843418860808015000
45	1.000000000000028400000000	0.000000000000028421709430404007000
46	1.000000000000014200000000	0.000000000000014210854715202004000
47	1.000000000000007100000000	0.000000000000007105427357601001900
48	1.000000000000003600000000	0.000000000000003552713678800500900
49	1.000000000000001800000000	0.000000000000001776356839400250500
50	1.000000000000000900000000	0.000000000000000888178419700125230
51	1.000000000000000400000000	0.000000000000000444089209850062620
52	1.000000000000000200000000	0.000000000000000222044604925031310
53	1.000000000000000000000000	0.000000000000000111022302462515650

El valor de ϵ es distinto de 10^{-308}, porque como estamos continuamente sumando 1 con ϵ, y ϵ se vuelve mas pequeño con cada iteración, y su magnitud es muy pequeña comparada con la de 1 y la suma de $1 + \epsilon$ deja de ser significativa.

Para precisión simple $\delta = 0{,}00097656250$ y para precisión doble $\delta = 0{,}000000000001818989403545856 50$.

Los valores de ϵ y δ son distintos ya que la magnitud de 10000 es mucho mayor a la de 1 y por lo tanto al sumarle números pequeños se llega de manera rápida a uno que no afecte la suma.

Respuesta 4

- Ascendente precisión simple:
- Ascendente precisión doble:
- Descendente precisión simple:

- Descendente precisión doble:
- Mayor a menor precisión simple:
- Mayor a menor precisión doble:
- Menor a mayor precisión simple:
- Menor a mayor precisión doble:

Respuesta 5

El mayor valor que llego a tomar la sumatoria fue **15.4036827008740234**. Fueron sumados **2097152 términos** antes de que la computadora dejara de "sumar".

La computadora no llega infinito al realizar la sumatoria debido a la precisión decimal, llega a un punto en que la computadora al sumar dos números, las magnitudes entre ellos son muy distintas y por lo tanto se queda con el numero mas grande y es como si no se le sumara nada.

Respuesta 6

Para $x = 10$, con precisión simple tenemos que $\boldsymbol{e^{10} = 22026{,}4667968750}$, este resultado se obtuvo al sumar los términos desde un $n = 0$, y hasta que la suma dejara de "sumar", usando al final **32 iteraciones**.

Y para precisión doble tenemos $\boldsymbol{e^{10} = 22026{,}465760913433769019320607185363769531250}$. que se obtuvo al cambiar el orden en que se suma y se empezó desde $n = 32$ hasta $n = 0$. En este caso usamos **32 iteraciones**.

Si se empezara desde un n muy grande y hasta $n = 0$, se tendría un resultado mas preciso.

Código Fuente

repuesta3.c

```c
#include <stdio.h>

void singlePrecision(float);
void doublePrecision(double);

int main(){
        singlePrecision(1);
        doublePrecision(1);
        singlePrecision(10000);
        doublePrecision(10000);
}

void singlePrecision(float x){
        float t = 2 * x, epsilon=1;
        int i=1;
        while(t > x){
                t = x + (epsilon *= 0.5);
                printf("i=%d t=%.24f epsilon=%.34f\n", i, t, epsilon);
                i++;
        }
}

void doublePrecision(double x){
        double t=2 * x, epsilon=1;
        int i=1;
        while(t > x){
                t = x + (epsilon *= 0.5);
                printf("i=%d t=%.24lf epsilon=%.34lf\n", i, t, epsilon);
                i++;
        }
}
```

repuesta4.c

```c
#include <stdio.h>

float productoEscalarAscendenteF(float[], float[]);
float productoEscalarDescendenteF(float[], float[]);

double productoEscalarAscendenteD(double[], double[]);
double productoEscalarDescendenteD(double[], double[]);

int main(){
        float a_float[] = {2.718281828, -3.141592654, 1.414213562,
                                        0.5772156649, 0.3010299957};

        float b_float[] = {1485.2497, 878366.9879, -22.37492,
                                        4773714.647, 0.000185049};

        double a_double[] = {2.718281828, -3.141592654, 1.414213562,
                                          0.5772156649, 0.3010299957};

        double b_double[] = {1485.2497, 878366.9879, -22.37492,
                                          4773714.647, 0.000185049};

        printf("Ascendiente Precision Simple   = %.34f\n", productoEscalarAscendenteF(a_float, b_float));
        printf("Descendiente Precision Simple  = %.34f\n", productoEscalarDescendenteF(a_float, b_float));
        printf("----\n");
        printf("Ascendiente Precision Doble    = %.34lf\n", productoEscalarAscendenteD(a_double, b_double));
        printf("Descendiente Precision Doble   = %.34lf\n", productoEscalarDescendenteD(a_double, b_double));
}

// funciones float
float productoEscalarAscendenteF(float a[], float b[]){
        int i;
        float producto=0;
        for(i=0; i < 5; i++){
                //~ printf("%.30f x %.30f\n", a[i], b[i]);
                producto += a[i] * b[i];
        }
        return producto;
}

float productoEscalarDescendenteF(float a[], float b[]){
        int i;
        float producto=0;
        for(i=4; i >= 0; i--){
                //~ printf("%.30f x %.30f\n", a[i], b[i]);
                producto += a[i] * b[i];
        }
        return producto;
}


// funciones double
double productoEscalarAscendenteD(double a[], double b[]){
```

```
53          int i;
54          double producto=0;
55          for(i=0; i < 5; i++){
56                  producto += a[i] * b[i];
57          }
58          return producto;
59  }
60
61  double productoEscalarDescendenteD(double a[], double b[]){
62          int i;
63          double producto=0;
64          for(i=4; i >= 0; i--){
65                  producto += a[i] * b[i];
66          }
67          return producto;
68  }
```

repuesta5.c

```c
#include <stdio.h>
#include <math.h>

void serieArmonicaSingle();

int main(){
        serieArmonicaSingle();
}

void serieArmonicaSingle(){
        float serie=0, ant=1;
        float k=1;

        while(serie - ant != 0){
                ant = serie;
                serie += 1 / (k++);
                //~ printf("serie: %.10f\n", serie);
        }

        printf("%f terminos.\n", k);
        printf("Serie = %.24f\n", serie);
}
```

repuesta6.c

```c
#include <stdio.h>
#include <math.h>

float exponencialAdelante(float);
float exponencialAtras(float);
float fact(float);
double exponencialAdelanteD(double);
double exponencialAtrasD(double);
double factD(double);

int main(){
        float x=10;
        printf("Precision Simple:\n");
        printf("\tHacia adelante = %.54f\n", exponencialAdelante(x));
        printf("\tHacia atras    = %.54f\n", exponencialAtras(x));
        printf("-------------------------------------------------------\n");
        printf("Precision Doble:\n");
        printf("\tHacia adelante = %.54lf\n", exponencialAdelanteD(x));
        printf("\tHacia atras    = %.54lf\n", exponencialAtrasD(x));


}

// Funciones float
float exponencialAdelante(float x){
        float e=0, factorial=0, potencia=0, ant=1, i=0;

        while(e != ant){
                factorial = factorial == 0 ? 1 : factorial * i;
                potencia = (i++) == 0 ? 1 : potencia * x;
                ant = e;
                e += potencia / factorial;
        }

        return e;
}

float exponencialAtras(float x){
        float e=0, factorial=0, potencia=0, n=33;

        while(n >= 0){
                factorial = fact(n);
                potencia = pow(x, (n--));
                e += potencia / factorial;
        }

        return e;
}

float fact(float n){
        int i;
        float f=1;
```

```
53          for(i=1; i <= n; i++){
54                  f *= i;
55          }
56          return f;
57  }
58
59
60  // Funciones double
61  double exponencialAdelanteD(double x){
62          double e=0, factorial=0, potencia=0, ant=1, i=0;
63
64          while(e != ant){
65                  factorial = (factorial == 0 ? 1 : factorial * i);
66                  potencia = ((i++) == 0 ? 1 : potencia * x);
67                  ant = e;
68                  e += potencia / factorial;
69          }
70
71          return e;
72  }
73
74  double exponencialAtrasD(double x){
75          double e=0, factorial=0, potencia=0, n=33;
76
77          while(n >= 0){
78                  factorial = fact(n);
79                  potencia = pow(x, (n--));
80                  e += potencia / factorial;
81          }
82
83          return e;
84  }
85
86  double factD(double n){
87          int i;
88          double f=1;
89          for(i=1; i <= n; i++){
90                  f *= i;
91          }
92          return f;
93  }
```