UNIVERSIDAD DE
GUANAJUATO

PRACTICA 1 CÁLCULO DE SERIES MEDIANTE ESTRUCTURAS DE CONTROL

POR: GUSTAVO GONZALEZ DE LA CRUZ, FERNANDO EMMANUEL TORRES AGUIRRE Y EMMANUEL VARGAS CISNEROS

13 DE MARZO DE 2025

MATERIA: PROGRAMACIÓN EN INGIENIERÍA

UNIVERSIDAD DE GUANAJUATO CAMPUS IRAPUATO-SALAMANCA

Contenido

Introducción

Como parte de nuestra primera práctica, se nos asignó la tarea de desarrollar diversos algoritmos destinados a resolver un conjunto de 33 ecuaciones correspondientes a las Series de Taylor. La resolución de estas ecuaciones debía lograrse mediante el uso de ciclos y sentencias de control en el lenguaje de programación C. Se estableció como variable de entrada tanto la cantidad de términos deseados para calcular el valor de como su valor correspondiente, permitiendo así obtener aproximaciones precisas para distintos valores de x.

Las Series de Taylor son fundamentales en el ámbito de las matemáticas y la ingeniería, ya que permiten aproximar funciones complejas mediante polinomios. Esta aproximación es especialmente útil cuando se requiere calcular valores de funciones que no pueden resolverse de forma exacta o directa, como las funciones exponenciales, logarítmicas y trigonométricas. Su relevancia radica en su aplicabilidad en múltiples campos, como la física, la ingeniería y la informática, facilitando el análisis y la resolución de problemas complejos a través de aproximaciones más manejables.

Durante el desarrollo de la práctica, nos enfrentamos a diversos desafíos y errores, desde complicaciones en la estructura de los algoritmos hasta dificultades en la interpretación de los resultados. Estos tropiezos iniciales sirvieron como base para el aprendizaje y la mejora continua, permitiéndonos perfeccionar cada algoritmo y afinar el proceso de cálculo. A continuación, se presentan los resultados obtenidos tras superar estos obstáculos:

Funciones Matemáticas

<table>
<tr><td>Función</td><td>Ln (2)</td></tr>
<tr><td>Descripción</td><td>El algoritmo calcula una aproximación de ln(2) utilizando una serie de sumas y restas que dependen del número de términos que se elijan. El número de términos es dado por un valor de entrada n. El algoritmo usa un ciclo For para recorrer desde 1 hasta n, y en cada iteración, agrega o resta un término de la serie. Al final del ciclo, el algoritmo devuelve una estimación del valor de ln(2). Cuantos más términos se usen, más precisa será la aproximación del valor.</td></tr>
<tr><td>Código</td><td>

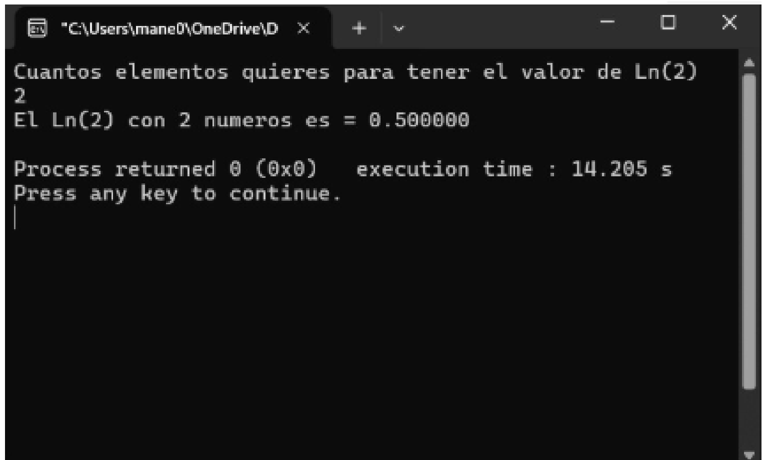
```c
int n = 0, i, d, sg;
    sg = 1; // Inicializa el signo como positivo
    d = 1; // Inicializa el divisor en 1
    float ln = 0.0; // Variable para almacenar el resultado de la aproximación de Ln(2)
    // Solicita el valor de n y asegura que sea mayor que 0
    do {
        printf("Cuantos elementos quieres para tener el valor de Ln(2)\n");
        scanf("%d", &n);
        if (n<=0) { // Verifica que la entrada sea válida y mayor que 0
            printf("Error: Entrada no valida. Ingrese un numero mayor o igual a 1\n");
            n = 0; // Reinicia n a 0 si la entrada no es válida
        }
    } while (n < 1); // Repite hasta que n sea válido
    // Bucle que calcula la serie para aproximar Ln(2)
    for (i = 0; i < n; i++, d++) {
```

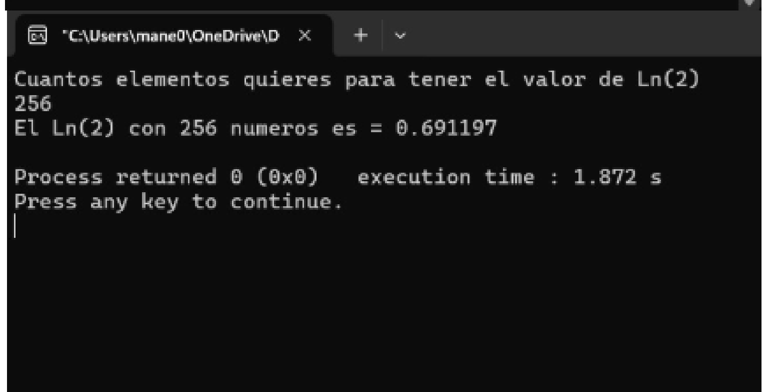
</td></tr>
</table>

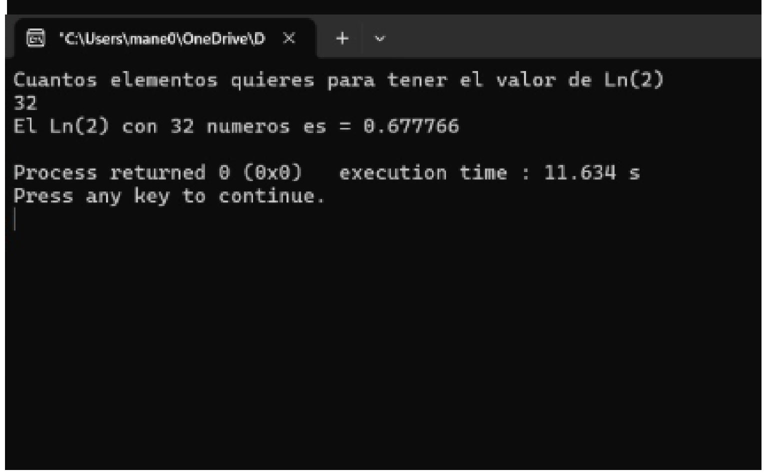
<table>
<tr>
<td></td>
<td>

```
        ln += sg * 1 / (float)(d); // Suma el término
correspondiente a la serie
        sg *= -1; // Alterna el signo entre positivo y
negativo
    }
    // Imprime el resultado final de Ln(2) con n términos
    printf("El Ln(2) con %d numeros es = %f\n", n, ln);
```

</td>
</tr>
<tr>
<td>Capturas</td>
<td>





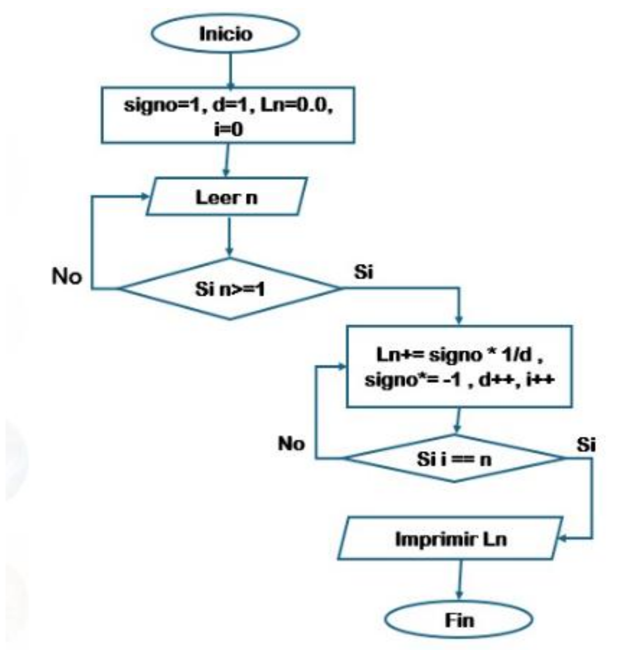


</td>
</tr>
</table>

Tabla de Comparación

n	x	Valor Aproximado	Error Relativo
2	No hay	0.5	27.8652
4	No hay	0.5833333	15.8428
8	No hay	0.634524	8.4575
16	No hay	0.662872	4.3678
32	No hay	0.677766	2.2190
64	No hay	0.685396	1.1183
128	No hay	0.689256	0.5614
256	No hay	0.691197	0.2814
		Ln(2)	0.6931

Diagrama de Flujo



Función

$$\frac{\pi}{4}$$

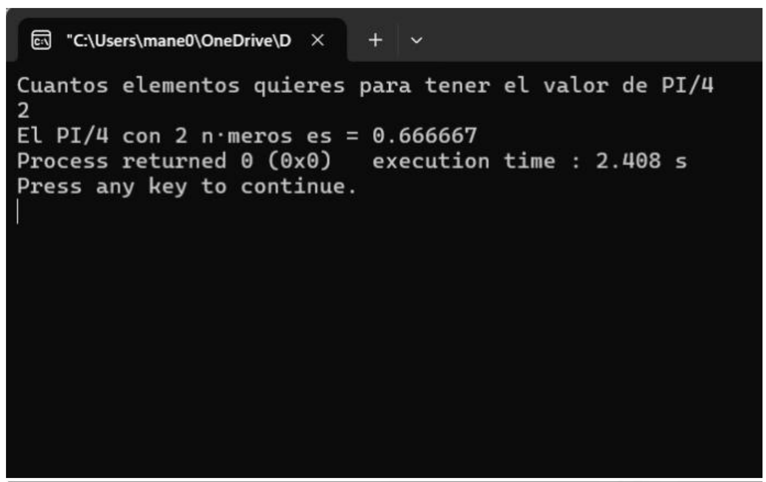
Descripción

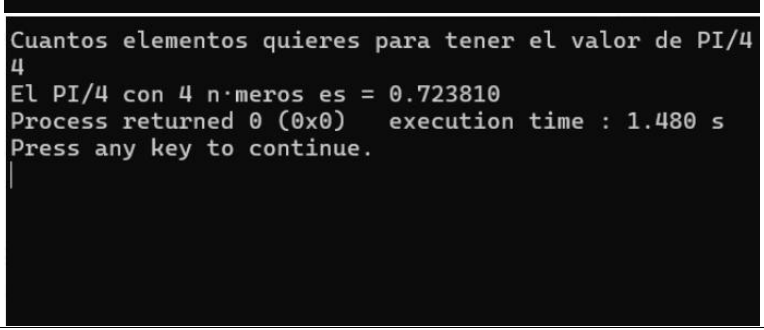
El algoritmo calcula una aproximación de Pi/4 utilizando una serie de sumas y restas que dependen del número de términos que se elijan. El número de términos es dado por un valor de entrada n. El algoritmo usa un ciclo For para recorrer desde 1 hasta n, y en cada iteración, agrega o resta un término de la serie. Al final del ciclo, el algoritmo devuelve una estimación del valor de Pi/4. Cuantos más términos se usen, más precisa será la aproximación del valor.

Código

```c
int n, i, d=1, s=1;
    float r = 0.0; // Resultado de la aproximación
    // Solicita el número de términos
    do {
        printf("Cuantos elementos quieres para tener el valor de PI/4\n");
        scanf("%d", &n); // Lee n
        if (n <= 0) { // Verifica si es válido
            printf("Error: Entrada no válida. Ingrese un número mayor o igual a 1\n");
            n = 0; // Reinicia n
        }
    } while (n <= 0); // Repite si no es válido
    // Calcula la serie para PI/4
    for (i = 0; i < n; i++) {
        r += s * 1 / (float)(d); // Suma el término
        s *= -1; // Cambia el signo
        d += 2; // Incrementa el divisor
    }
    // Muestra el resultado
    printf("El PI/4 con %d números es = %f", n, r);
```

Capturas





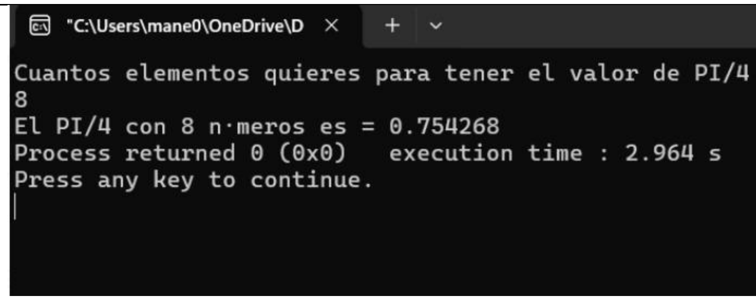
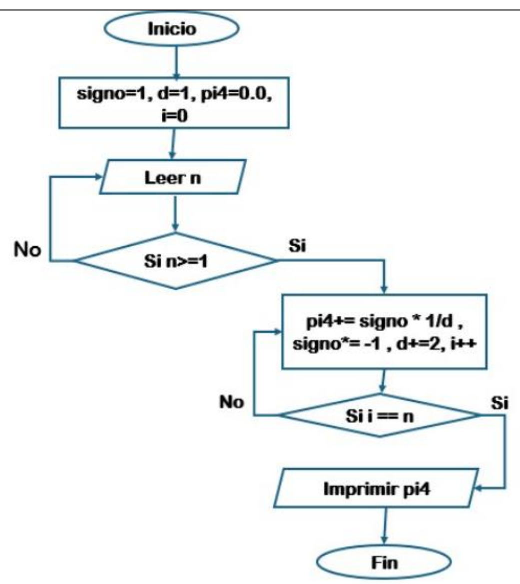


Tabla de Comparación

n	x	Valor Aproximado	Error Relativo
2	No hay	0.66667	15.1169
4	No hay	0.72381	7.8416
8	No hay	0.754268	3.9636
16	No hay	0.769788	1.9875
32	No hay	0.777587	0.9945
64	No hay	0.781492	0.4973
128	No hay	0.783445	0.2487
256	No hay	0.784422	0.1243
		Pi/4	0.7854

Diagrama



Función

$$\frac{\pi^2}{6}$$

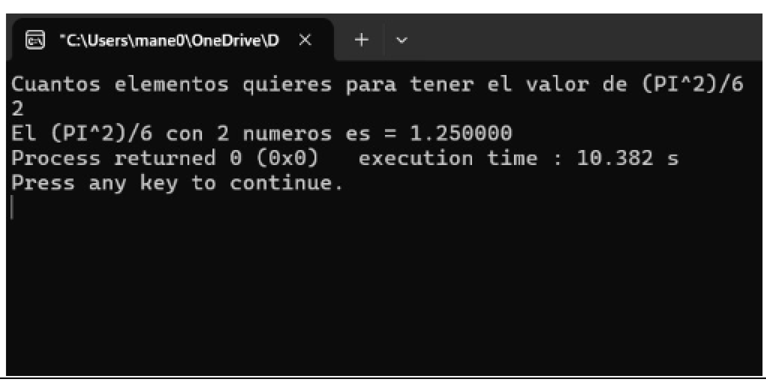
Descripción

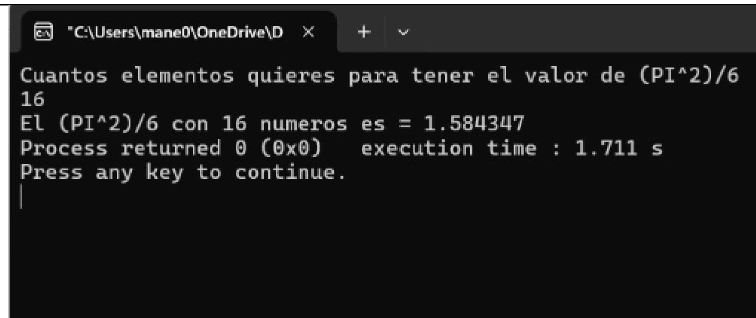
El algoritmo calcula una aproximación de (Pi^2)/6 utilizando una serie de sumas y restas que dependen del número de términos que se elijan. El número de términos es dado por un valor de entrada n. El algoritmo usa un ciclo For para recorrer desde 1 hasta n, y en cada iteración, agrega o resta un término de la serie. Al final del ciclo, el algoritmo devuelve una estimación del valor de (Pi^2)/6-
Cuantos más términos se usen, más precisa será la aproximación del valor.

Código

```
int n, i;
    float pi26 = 0.0; // Resultado de la aproximación
    do {
        printf("Cuantos elementos quieres para tener el valor de (PI^2)/6 \n");
        scanf("%d", &n); // Lee n
        if (n <= 0) { // Verifica si es válido
            printf("Error: Entrada no válida. Ingrese un número mayor o igual a 1\n");
            n = 0; // Reinicia n
        }
    } while (n < 1); // Repite si n es menor que 1

    // Calcula la serie para (PI^2)/6
    for (i = 1; i <= n; i++) {
        pi26 += 1 / (float)(i * i); // Suma el término correspondiente
    }

    // Muestra el resultado
    printf("El (PI^2)/6 con %d numeros es = %f", n, pi26);
```

Capturas





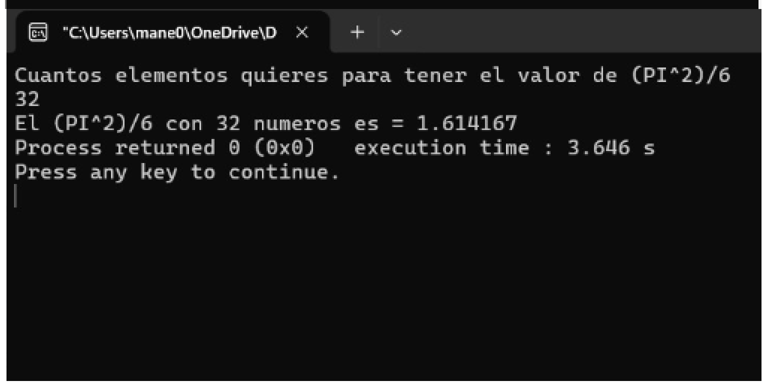
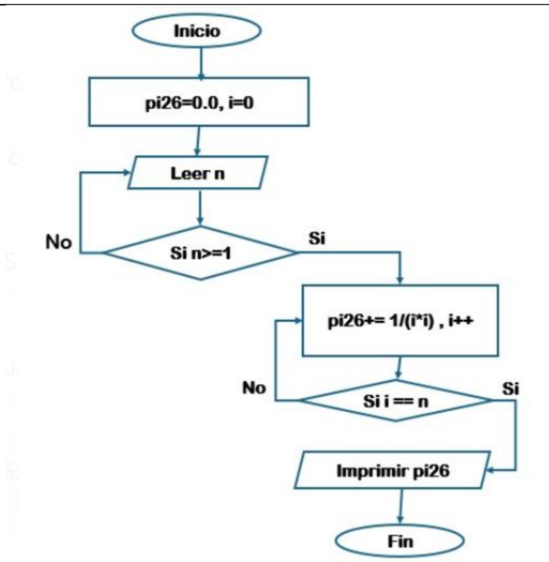


Tabla de Comparación

n	x	Valor Aproximado	Error Relativo
2	No hay	1.25	24.0091
4	No hay	1.423611	13.4548
8	No hay	1.527422	7.1439
16	No hay	1.584347	3.6833
32	No hay	1.614167	1.8704
64	No hay	1.629431	0.9425
128	No hay	1.637152	0.4731
256	No hay	1.641036	0.2370
		Pi2/6	1.6449

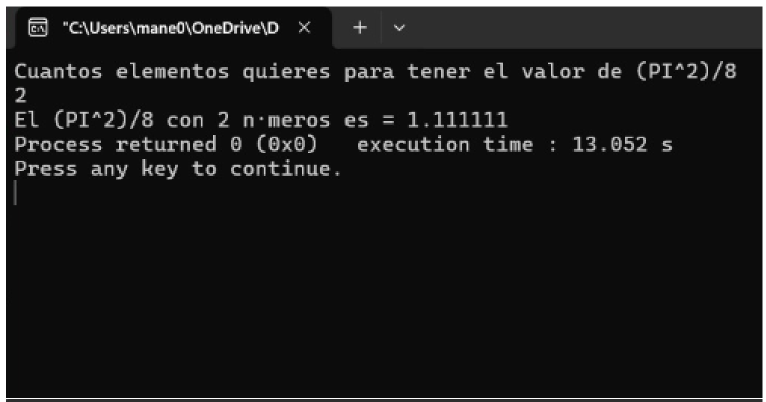
Diagrama

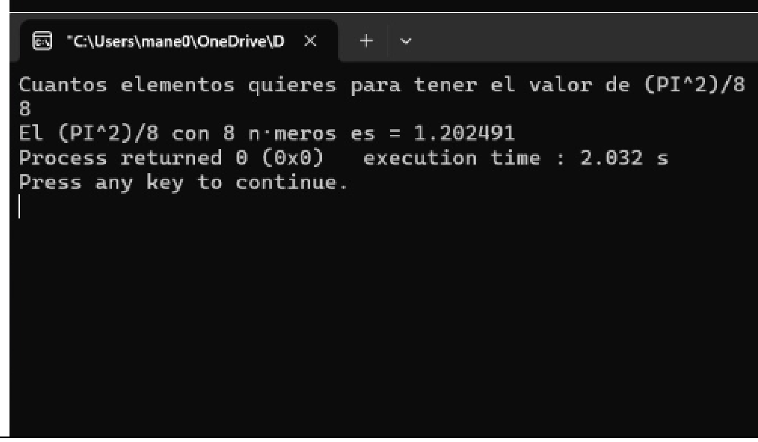
	

Función	$\frac{\pi^2}{8}$
Descripción	El algoritmo calcula una aproximación de (Pi^2)/8 utilizando una serie de sumas y restas que dependen del número de términos que se elijan. El número de términos es dado por un valor de entrada n. El algoritmo usa un ciclo For para recorrer desde 1 hasta n, y en cada iteración, agrega o resta un término de la serie. Al final del ciclo, el algoritmo devuelve una estimación del valor de (Pi^2)/8 Cuantos más términos se usen, más precisa será la aproximación del valor.
Código	`int n, i, d; // n: número de términos, i: iterador, d: divisor` `    d = 1; // Divisor inicial` `    float pi28 = 0.0; // Resultado de la aproximación` `    // Solicita el número de términos` `    do {` `        printf("Cuantos elementos quieres para tener el valor de (PI^2)/8\n");` `        scanf("%d", &n); // Lee n` `        if (n <= 0) { // Verifica si es válido` `            printf("Error: Entrada no válida. Ingrese un número mayor o igual a 1\n");`

<table>
<tr><td></td><td><pre>
            n = 0; // Reinicia n
        }
    } while (n < 1); // Repite si n es menor que 1

    // Calcula la serie para (PI^2)/8
    for (i = 0; i < n; i++) {
        pi28 += 1 / (float)(d * d); // Suma el término
        d += 2; // Incrementa el divisor
    }

    // Muestra el resultado
    printf("El (PI^2)/8 con %d números es = %f", n,
pi28);
</pre></td></tr>
<tr><td>Capturas</td><td>




</td></tr>
</table>

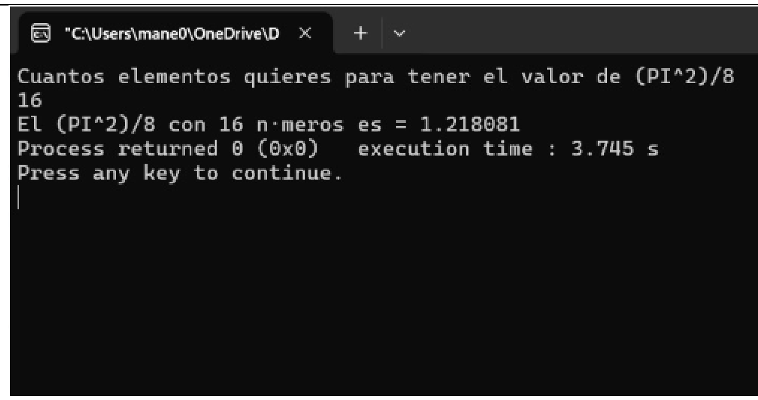
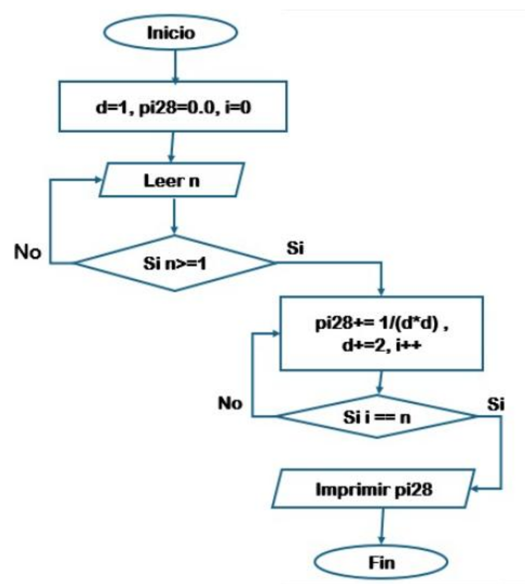


Tabla de Comparación

n	x	Valor Aproximado	Error Relativo
2	No hay	1.111111	9.9367
4	No hay	1.171519	5.0402
8	No hay	1.202491	2.5298
16	No hay	1.218081	1.2661
32	No hay	1.225889	0.6332
64	No hay	1.229794	0.3167
128	No hay	1.231748	0.1583
256	No hay	1.232724	0.0792
		Pi2/8	1.2337

Diagrama



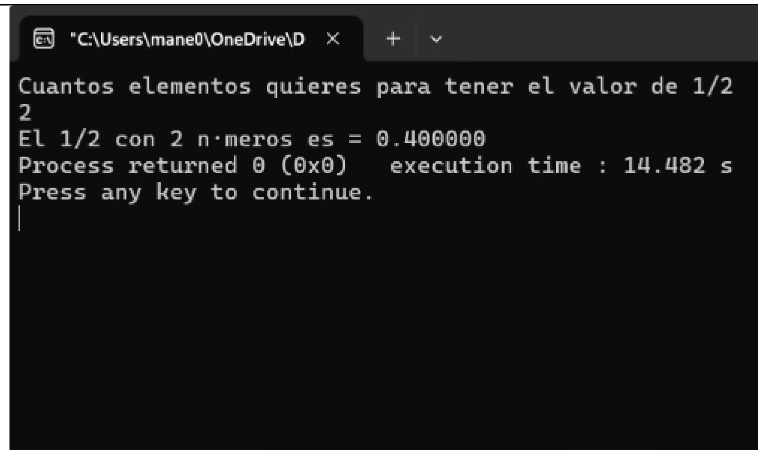
<table>
<tr><td>Función</td><td>

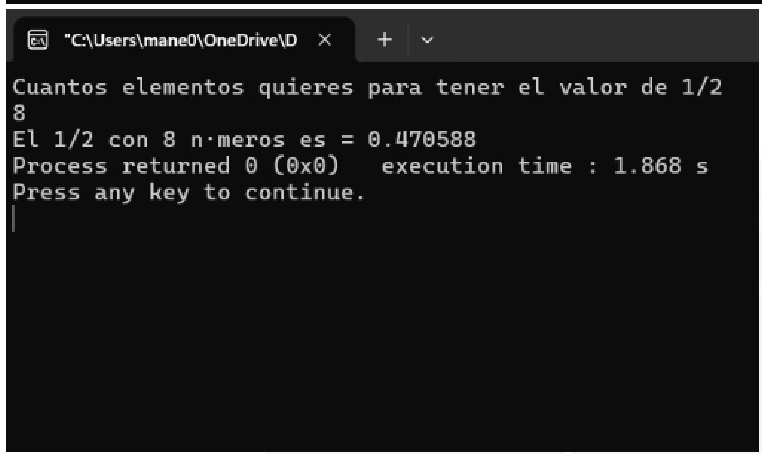
$$\frac{1}{2}$$

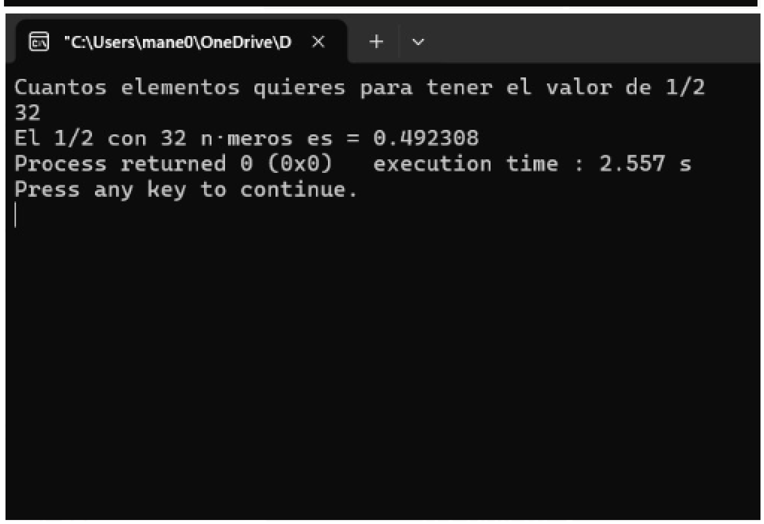
</td></tr>
<tr><td>Descripción</td><td>El algoritmo calcula una aproximación de 1/2 utilizando una serie de sumas y restas que dependen del número de términos que se elijan. El número de términos es dado por un valor de entrada n. El algoritmo usa un ciclo For para recorrer desde 1 hasta n, y en cada iteración, agrega o resta un término de la serie. Al final del ciclo, el algoritmo devuelve una estimación del valor de 1/2 Cuantos más términos se usen, más precisa será la aproximación del valor.</td></tr>
<tr><td>Código</td><td>

```
int n, i, d;
    d = 1; // Divisor inicial
    float med = 0.0; // Resultado de la suma

    // Solicita el número de términos para la serie
    do {
        printf("Cuantos elementos quieres para tener el valor de 1/2\n");
        scanf("%d", &n); // Lee el valor de n
        if (n <= 0) { // Verifica si n es válido
            printf("Error: Entrada no válida. Ingrese un número mayor o igual a 1\n");
            n = 0; // Reinicia n
        }
    } while (n < 1); // Repite si n es menor que 1

    // Calcula la serie para 1/2
    for (i = 0; i < n; i++) {
        med += 1 / (float)(d * (d + 2)); // Suma el término de la serie
        d += 2; // Incrementa el divisor por 2
    }

    // Muestra el resultado
    printf("El 1/2 con %d números es = %f", n, med);
```

</td></tr>
<tr><td>Capturas</td><td></td></tr>
</table>

<table>
<tr><td></td><td>


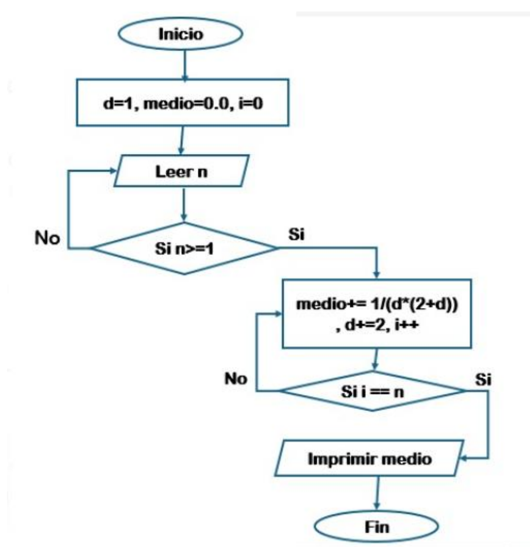




</td></tr>
<tr><td>Tabla de Comparación</td><td></td></tr>
</table>

n	x	Valor Aproximado	Error Relativo
2	No hay	0.4	20.00
4	No hay	0.44444	11.11
8	No hay	0.470588	5.88
16	No hay	0.484848	3.03
32	No hay	0.492308	1.54
64	No hay	0.496124	0.78
128	No hay	0.498054	0.39
256	No hay	0.499025	0.20
		1/2	0.50

Diagrama

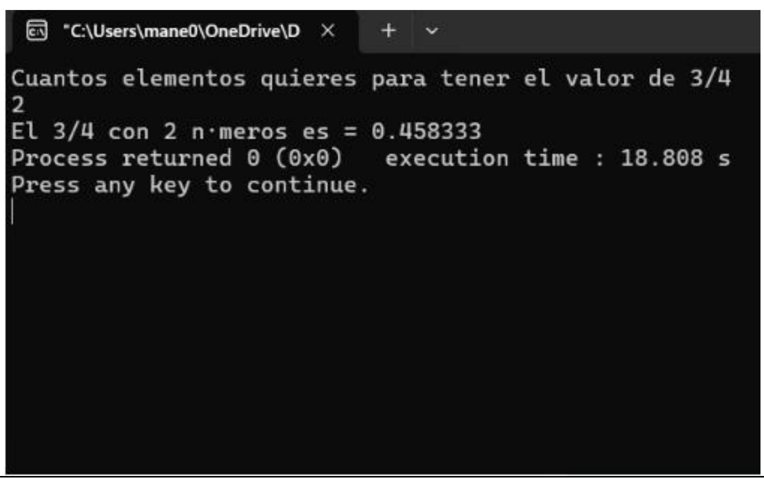


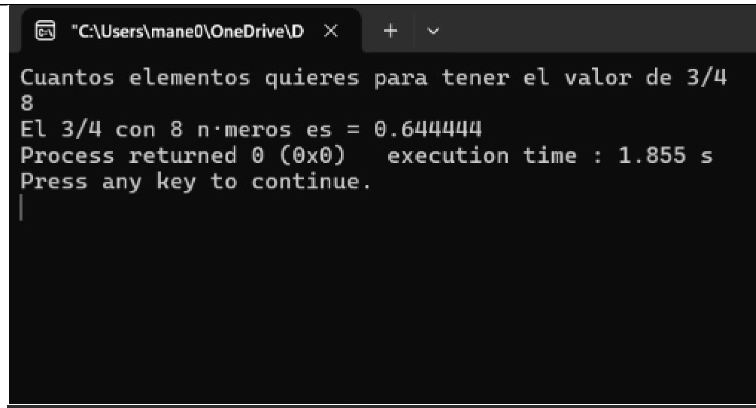
Función	$$\frac{3}{4}$$
Descripción	El algoritmo calcula una aproximación de 3/4 utilizando una serie de sumas y restas que dependen del número de términos que se elijan. El número de términos es dado por un valor de entrada n. El algoritmo usa un ciclo For para recorrer desde 1 hasta n, y en cada iteración, agrega o resta un término de la serie. Al final del ciclo, el algoritmo devuelve una estimación del valor de 3/4 Cuantos más términos se usen, más precisa será la aproximación del valor.

Código

```
int n, i, d;
    d = 1; // Divisor inicial
    float trecua = 0.0; // Resultado de la serie

    // Solicita el número de términos para calcular el
valor de 3/4
    do {
        printf("Cuantos elementos quieres para tener el
valor de 3/4\n");
        scanf("%d", &n); // Lee el valor de n
        if (n <= 0) { // Verifica si n es válido
            printf("Error: Entrada no válida. Ingrese un
número mayor o igual a 1\n");
            n = 0; // Reinicia n
        }
    } while (n < 1); // Repite si n es menor que 1

    // Calcula la serie para 3/4
    for (i = 1; i <= n; i++) {
        trecua += 1 / (float)(i * (i + 2)); // Suma el
término correspondiente a la serie
    }

    // Muestra el resultado
    printf("El 3/4 con %d números es = %f", n, trecua);
```

Capturas





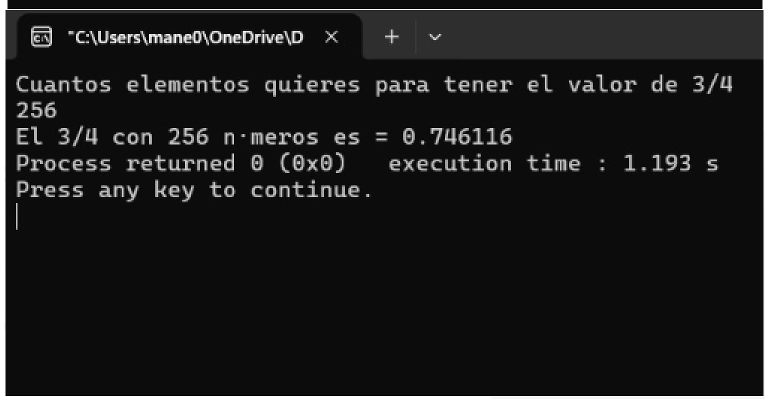
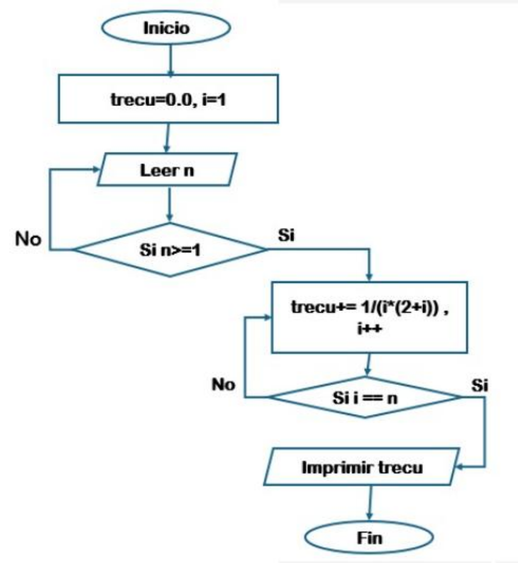


Tabla de Comparación

n	x	Valor Aproximado	Error Relativo
2	No hay	0.458333	38.89
4	No hay	0.56666	24.45
8	No hay	0.644444	14.07
16	No hay	0.69281	7.63
32	No hay	0.720143	3.98
64	No hay	0.734732	2.04
128	No hay	0.742278	1.03
256	No hay	0.746116	0.52
		3/4	0.75

Diagrama

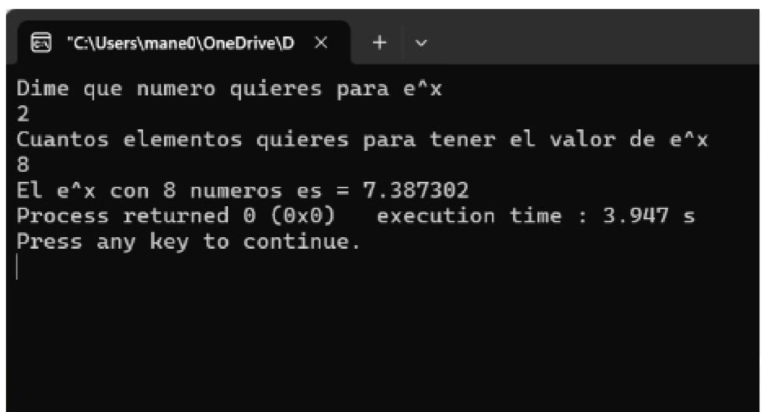
	

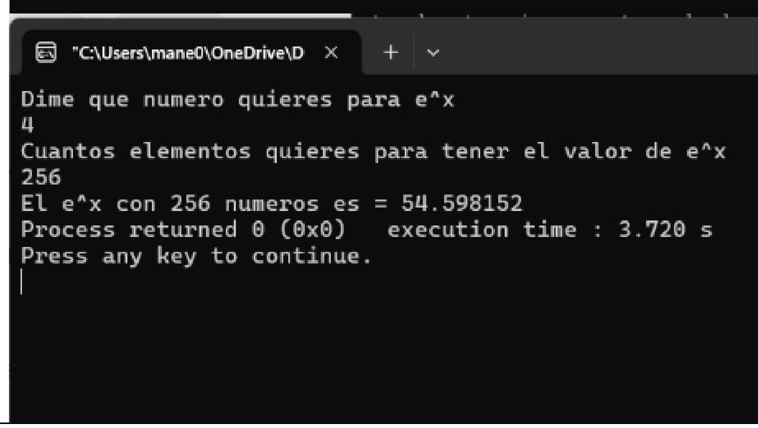
Funciones Exponenciales y Logarítmicas

Función	e^x
Descripción	El algoritmo calcula una aproximación de e^x utilizando una serie similar, pero en este caso, el valor de x es definido por el usuario. El número de términos de la serie sigue siendo determinado por n, el cual también es ingresado por el usuario. Dentro de un ciclo For, el algoritmo suma o resta los términos de la serie para calcular la aproximación de e^x. En cada iteración, se usa el valor de x proporcionado para ajustar el término que se agrega o resta. Al final del ciclo, el algoritmo devuelve el valor aproximado de e^x, y cuanto más grande sea el número de términos, más precisa será la estimación.
Código	int n, i, d, x; d = 1; float ex = 1.0, fact = 1.0; // Solicita el valor de x printf("Dime que numero quieres para e^x\n"); scanf("%d", &x); // Solicita el número de términos n para la aproximación de e^x do {

```
        printf("Cuantos elementos quieres para tener el valor de e^x\n");
        scanf("%d", &n); // Lee el número de términos (n)
        if (n <= 0) { // Verifica si n es válido
            printf("Error: Entrada no válida. Ingrese un número mayor o igual a 1\n");
            n = 0; // Reinicia n si la entrada no es válida
        }
    } while (n < 1); // Repite si n es menor que 1
    // Calcula la serie para e^x
    for (i = 1; i <= n; i++) {
        fact = 1; // Reinicia el valor del factorial
        // Calcula el factorial para el término i de la serie
        for (int j = 1; j <= i; j++) {
            fact *= x / (float)j;
        }
        ex += fact; // Suma el término calculado a e^x
    }
    // Imprime el resultado final de e^x con n términos
    printf("El e^x con %d numeros es = %f", n, ex);
```

Capturas





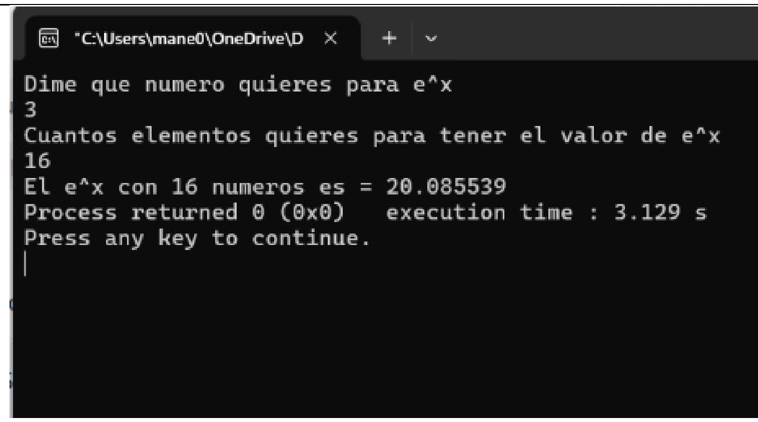
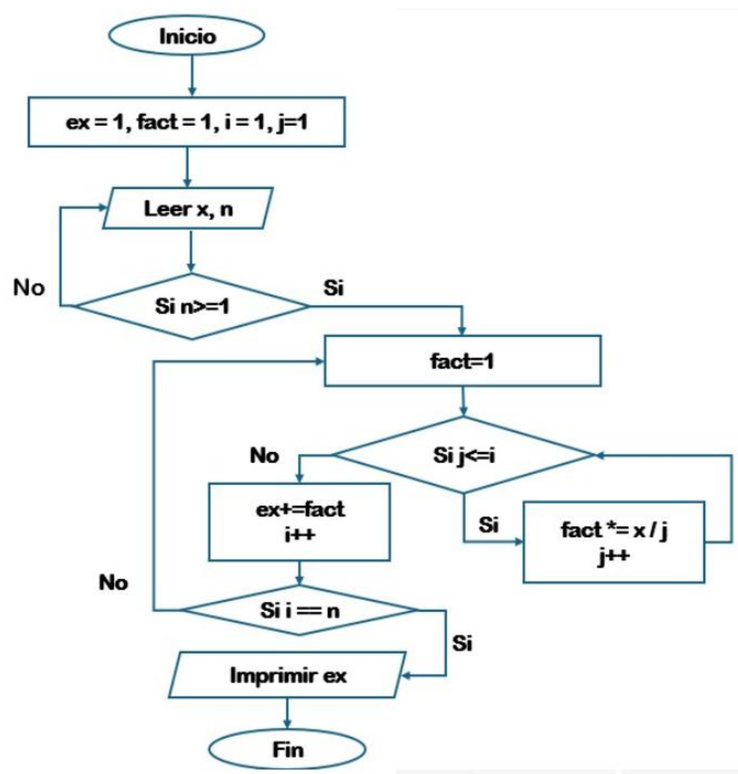


Tabla de Comparación

	e^2		e^4		e^3	
n	Valor	Error	Valor	Error	Valor	Error
2	5	0.3233	13	76.190	8.5	57.681
4	7	0.0553	34.33333	37.116	16.375	18.47368
8	7.387302	0.0002	53.43175	2.136	20.00915	0.380303
16	7.389057	0.0000	54.59815	0.000	20.08554	0.00
32	7.389057	0.0000	54.59815	0.000	20.08554	0.00
64	7.389057	0.0000	54.59815	0.000	20.08554	0.00
128	7.389057	0.0000	54.59815	0.000	20.08554	0.00
256	7.389057	0.0000	54.59815	0.000	20.08554	0.00

Diagrama



<table>
<tr><td>Función</td><td>xe^x</td></tr>
<tr><td>Descripción</td><td>El algoritmo calcula una aproximación de xe^x utilizando una serie similar, pero en este caso, el valor de x es definido por el usuario. El número de términos de la serie sigue siendo determinado por n, el cual también es ingresado por el usuario.
Dentro de un ciclo For, el algoritmo suma o resta los términos de la serie para calcular la aproximación de xe^x. En cada iteración, se usa el valor de x proporcionado para ajustar el término que se agrega o resta. Al final del ciclo, el algoritmo devuelve el valor aproximado de xe^x, y cuanto más grande sea el número de términos, más precisa será la estimación.</td></tr>
<tr><td>Código</td><td><pre>
int n, i, d, x; // n: número de términos, x: valor para e^x
    d = 1; // Inicializa el divisor en 1
    float ex = 0.0, fact = 1.0; // ex: resultado final, fact: factorial

    printf("Dime que numero quieres para xe^x\n");
    scanf("%d", &x); // Lee el valor de x

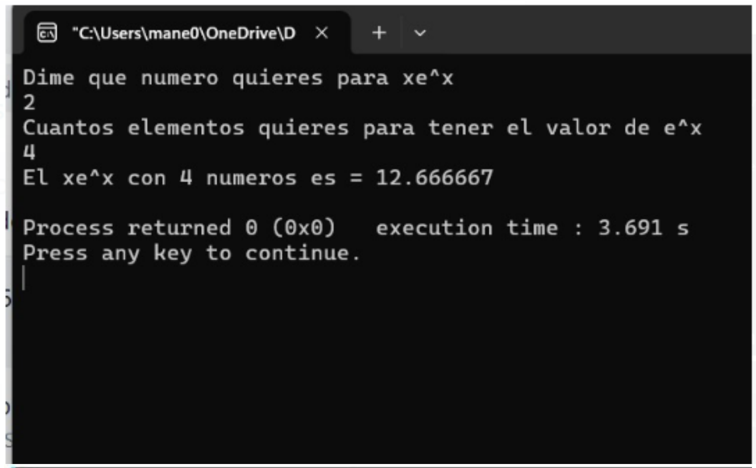
    // Solicita el número de términos
    do {
        printf("Cuantos elementos quieres para tener el valor de e^x\n");
        scanf("%d", &n); // Lee el valor de n
        if (n <= 0) { // Verifica que n sea mayor que 0
            printf("Error: Entrada no válida. Ingrese un número mayor o igual a 1\n");
            n = 0; // Reinicia n si la entrada es incorrecta
        }
    } while (n < 1); // Repite si n es menor que 1

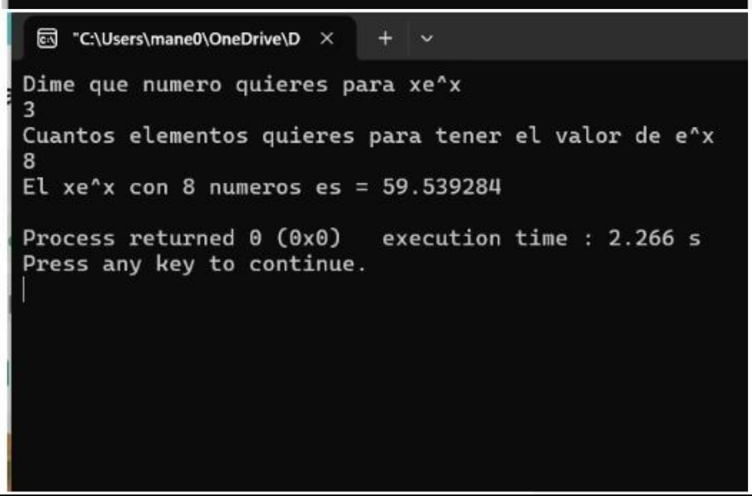
    // Bucle que calcula la serie para xe^x
    for (i = 1; i <= n; i++) {
        fact = 1; // Inicializa el factorial en 1
        // Bucle que calcula el término del factorial
        for (int j = 1; j <= i; j++) {
            fact *= x / (float)j; // Calcula el término del factorial
        }
        fact *= i; // Multiplica el término por i
</pre></td></tr>
</table>

<table>
<tr><td></td><td>

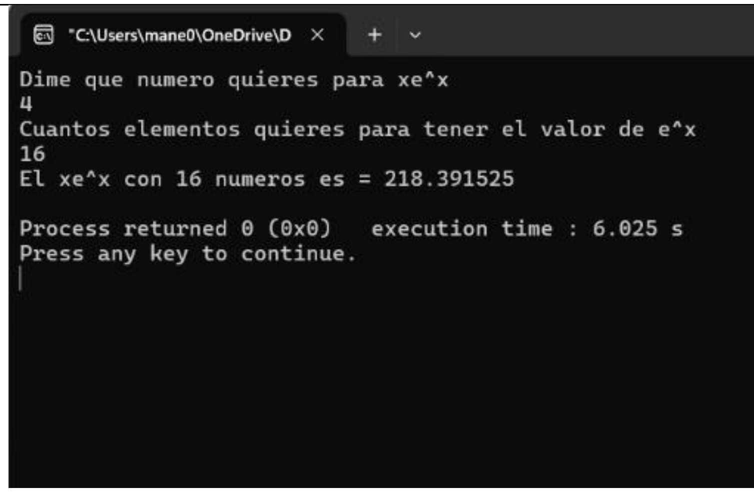
```
        ex += fact; // Suma el término calculado al
resultado final
    }

    // Muestra el resultado de xe^x con n términos
    printf("El xe^x con %d numeros es = %f\n", n, ex);
```

</td></tr>
<tr><td>Capturas</td><td>





</td></tr>
</table>

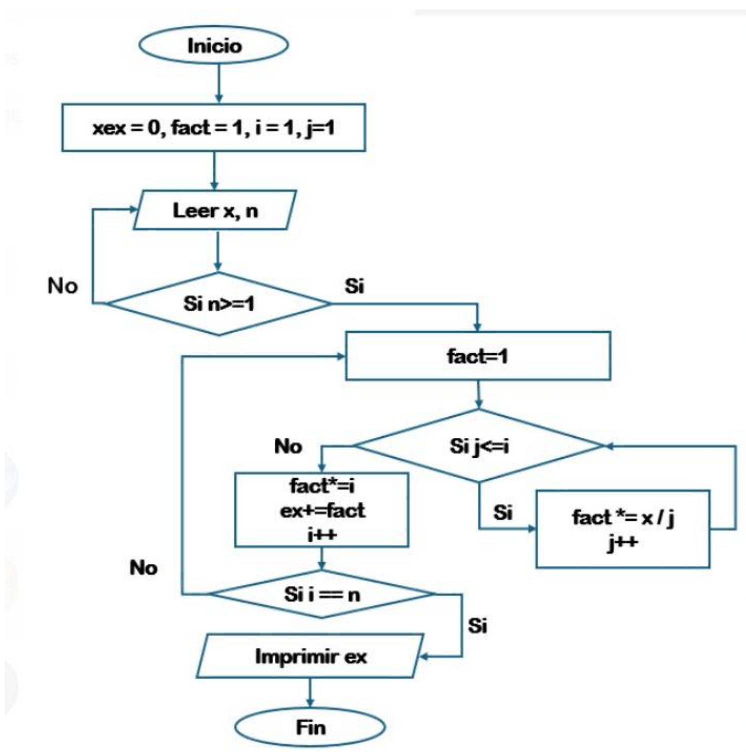


```
"C:\Users\mane0\OneDrive\D

Dime que numero quieres para xe^x
4
Cuantos elementos quieres para tener el valor de e^x
16
El xe^x con 16 numeros es = 218.391525

Process returned 0 (0x0)   execution time : 6.025 s
Press any key to continue.
```

Tabla de Comparación

	2e^2		3e^3		4e^4	
n	Valor	Error	Valor	Error	Valor	Error
2	6	59.3994	12	80.085	20	90.84218
4	12.66667	14.2876	39	35.277	94.66667	56.65299
8	14.76191	0.1097	59.53929	1.190	207.2254	5.113363
16	14.77811	0.0000	60.25662	0.000	218.3915	0.000496
32	14.77811	0.0000	60.25662	0.000	218.3926	0.00
64	14.77811	0.0000	60.25662	0.000	218.3926	0.00
128	14.77811	0.0000	60.25662	0.000	218.3926	0.00
256	14.77811	0.0000	60.25662	0.000	218.3926	0.00

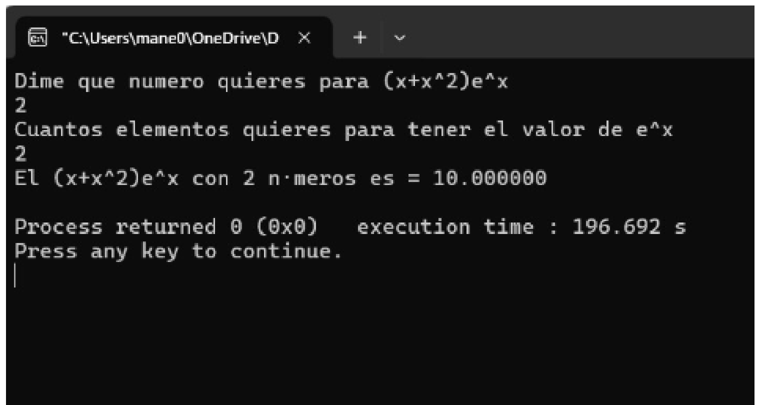
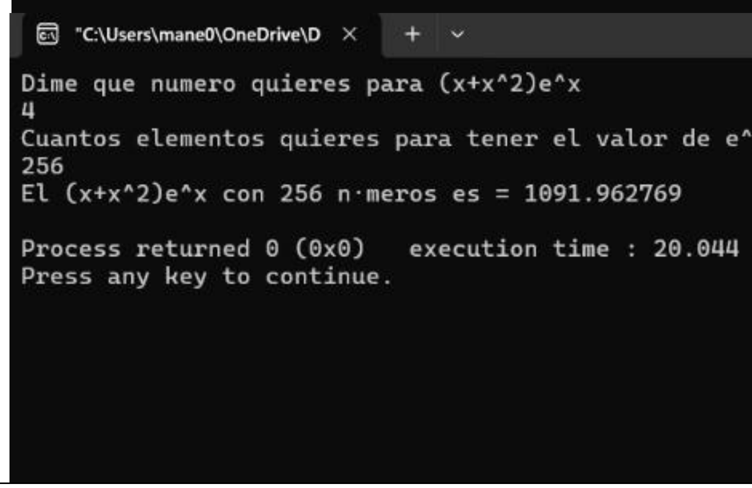
Diagrama

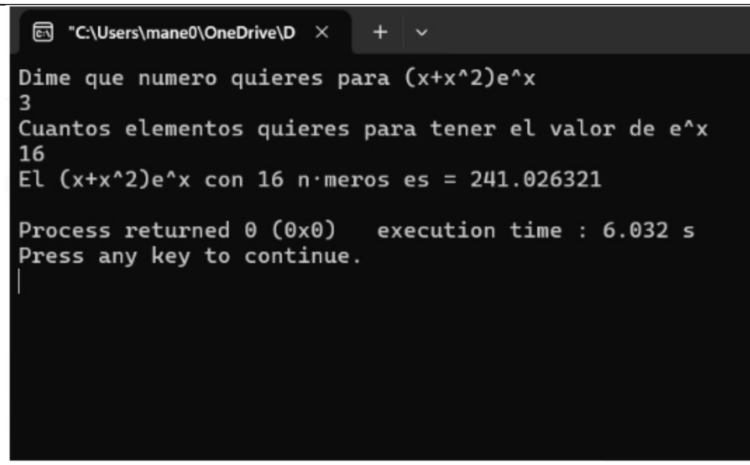


Función	$(x + x^2)e^x$
Descripción	El algoritmo calcula una aproximación de (x+x^2)e^x utilizando una serie similar, pero en este caso, el valor de x es definido por el usuario. El número de términos de la serie sigue siendo determinado por n, el cual también es ingresado por el usuario. Dentro de un ciclo For, el algoritmo suma o resta los términos de la serie para calcular la aproximación de (x+x^2)e^x. En cada iteración, se usa el valor de x proporcionado para ajustar el término que se agrega o resta. Al final del ciclo, el algoritmo devuelve el valor aproximado de (x+x^2)e^x, y cuanto más grande sea el número de términos, más precisa será la estimación.

Código

```
int n, i, d, x;
    d = 1; // Inicializa el divisor en 1
    float ex = 0.0, fact = 1.0; // ex: resultado final,
fact: factorial
    // Solicita el valor de x para la expresión (x +
x^2)e^x
    printf("Dime que numero quieres para (x+x^2)e^x\n");
    scanf("%d", &x); // Lee el valor de x
    // Solicita el número de términos para la
aproximación
    do {
        printf("Cuantos elementos quieres para tener el
valor de e^x\n");
        scanf("%d", &n); // Lee el valor de n
        if (n <= 0) { // Verifica si n es válido (mayor
que 0)
            printf("Error: Entrada no válida. Ingrese un
número mayor o igual a 1\n");
            n = 0; // Reinicia n si la entrada es
incorrecta
        }
    } while (n < 1); // Repite si n es menor que 1

    // Bucle que calcula la serie para (x + x^2)e^x
    for (i = 1; i <= n; i++) {
        fact = 1; // Inicializa el factorial en 1
        // Bucle que calcula el término del factorial
        for (int j = 1; j <= i; j++) {
            fact *= x / (float)j; // Calcula el término
del factorial
        }
```

	fact *= (i * i); // Multiplica el término por (i^2) como parte de la expresión ex += fact; // Suma el término calculado al resultado final } // Muestra el resultado de (x + x^2)e^x con n términos printf("El (x+x^2)e^x con %d números es = %f\n", n, ex);
Capturas	

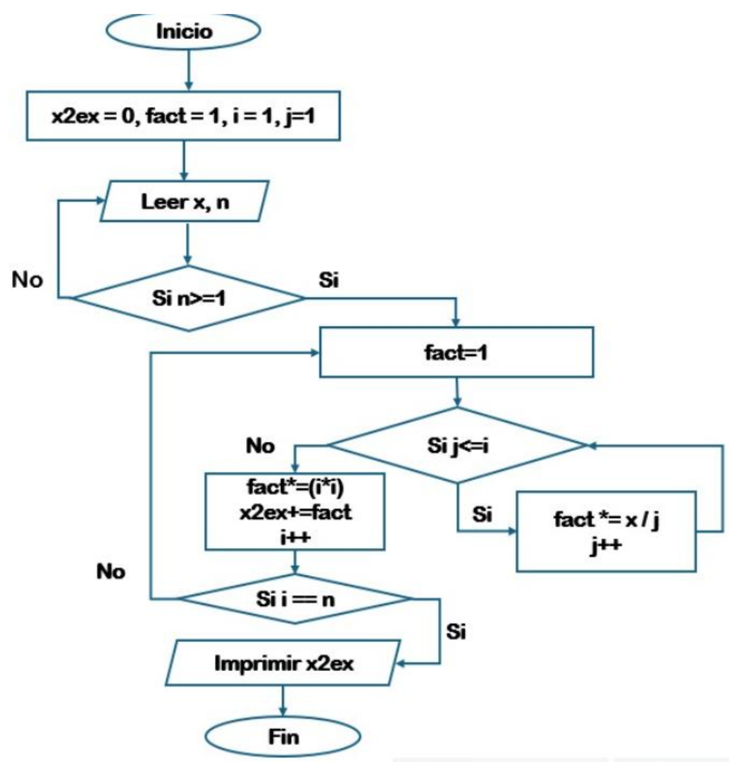


```
"C:\Users\mane0\OneDrive\D
Dime que numero quieres para (x+x^2)e^x
3
Cuantos elementos quieres para tener el valor de e^x
16
El (x+x^2)e^x con 16 n·meros es = 241.026321

Process returned 0 (0x0)   execution time : 6.032 s
Press any key to continue.
```

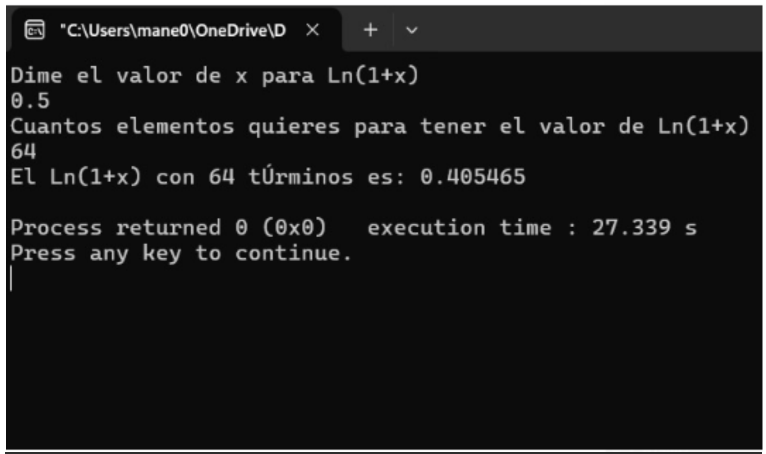
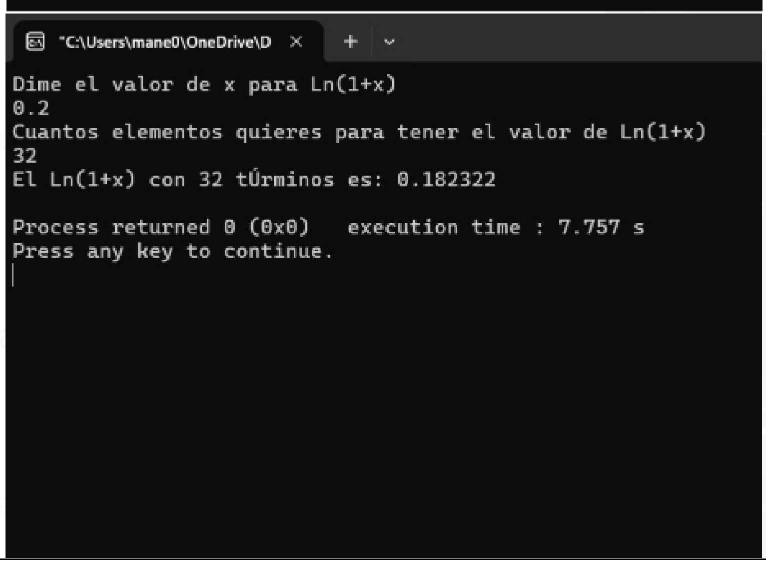
Tabla de Comparación

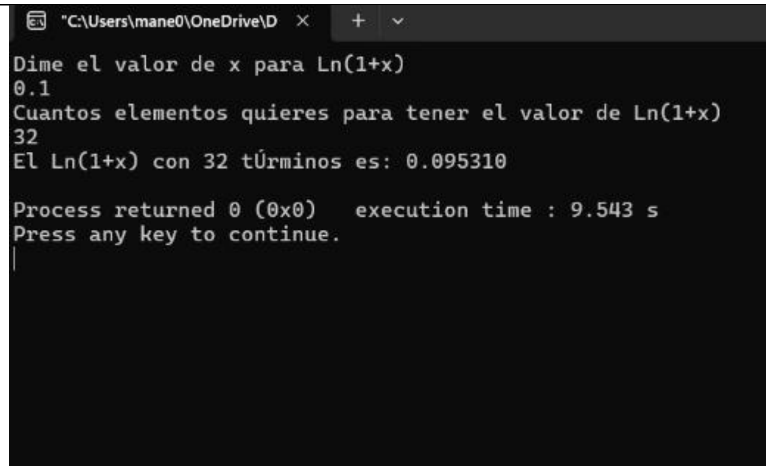
	$(2+2^2)e^2$		$(3+3^2)e^3$		$(4+4^2)e^4$	
n	Valor	Error	Valor	Error	Valor	Error
2	10	77.4441	21	91.287	36	96.70318
4	32.6666	26.3176	115.5	52.080	302.6669	72.28231
8	44.18413	0.3388	234.2518	2.811	984.114	9.876608
16	44.33434	0.0000	241.0263	0.000	1091.945	0.001678
32	44.33434	0.0000	241.0264	0.000	1091.963	0.000012
64	44.33434	0.0000	241.0264	0.000	1091.963	0.000012
128	44.33434	0.0000	241.0264	0.000	1091.963	0.000012
256	44.33434	0.0000	241.0264	0.000	1091.963	0.000012

Diagrama



Función	Ln(1 + x)
Descripción	El algoritmo calcula una aproximación de Ln(1+x) utilizando una serie similar, pero en este caso, el valor de x es definido por el usuario. El número de términos de la serie sigue siendo determinado por n, el cual también es ingresado por el usuario. Dentro de un ciclo For, el algoritmo suma o resta los términos de la serie para calcular la aproximación de Ln(1+x). En cada iteración, se usa el valor de x proporcionado para ajustar el término que se agrega o resta. Al final del ciclo, el algoritmo devuelve el valor aproximado de Ln(1+x), y cuanto más grande sea el número de términos, más precisa será la estimación.
Código	int n, i, sig = 1; // n: número de términos, i: iterador, sig: signo alternante float x, ln = 0.0, term; // x: valor de entrada para la función Ln(1+x), ln: resultado de la aproximación, term: término de la serie // Solicita el valor de x para la función Ln(1+x), asegurándose de que esté en el rango (-1, 1) do { printf("Dime el valor de x para Ln(1+x)\n"); scanf("%f", &x); // Lee el valor de x } while (x <= -1 || x >= 1); // Repite si x está fuera del rango (-1, 1) // Solicita el número de términos para la aproximación de Ln(1+x) do { printf("Cuantos elementos quieres para tener el valor de Ln(1+x)\n"); scanf("%d", &n); // Lee el número de términos if (n <= 0) { // Verifica si n es válido printf("Error: Entrada no válida. Ingrese un número mayor o igual a 1\n"); n = 0; // Reinicia n si la entrada es incorrecta } } while (n < 1); // Repite si n es menor que 1 // Calcula la serie para Ln(1+x) utilizando n términos for (i = 1; i <= n; i++) { term = 1; // Inicializa el término // Calcula el término x^i for (int j = 1; j <= i; j++) { term *= x; // Calcula la potencia de x

	`}` `// Agrega el término calculado, alternando el signo` `ln += sig * term / i;` `sig *= -1; // Alterna el signo para el siguiente término` `}` `// Muestra el resultado de la aproximación de Ln(1+x) con n términos` `printf("El Ln(1+x) con %d términos es: %f\n", n, ln);`
Capturas	 

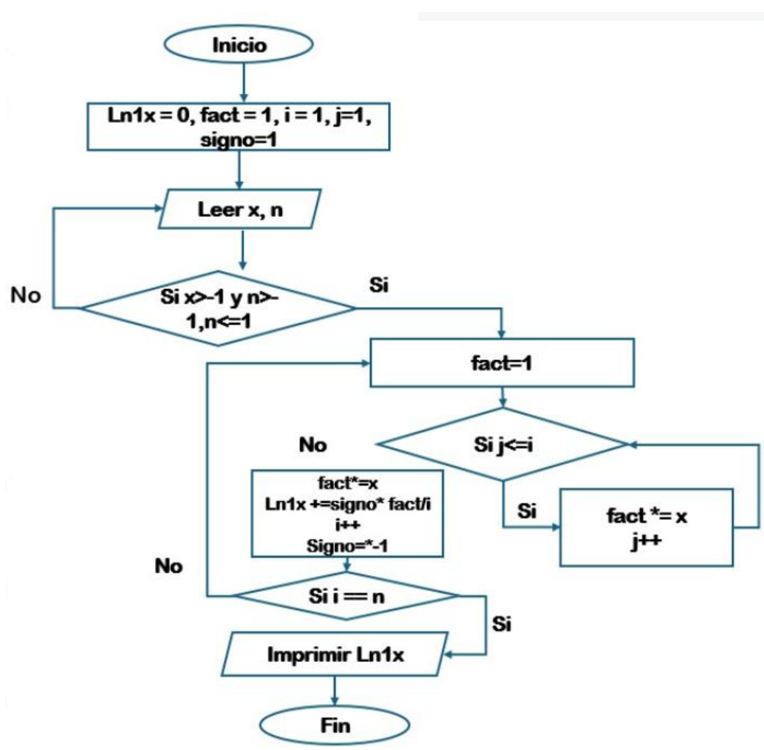


```
"C:\Users\mane0\OneDrive\D
Dime el valor de x para Ln(1+x)
0.1
Cuantos elementos quieres para tener el valor de Ln(1+x)
32
El Ln(1+x) con 32 tÚrminos es: 0.095310

Process returned 0 (0x0)   execution time : 9.543 s
Press any key to continue.
```

Tabla de Comparación

	Ln(1+0.1)		Ln(1+0.2)		Ln(1+0.5)	
n	Valor	Error	Valor	Error	Valor	Error
2	0.095	0.3253	0.18	1.274	0.375	7.5135955
4	0.095308	0.0021	0.182267	0.030	0.401042	1.0908463
8	0.09531	0.0000	0.182322	0.000	0.405315	0.0369946
16	0.09531	0.0000	0.182322	0.000	0.405465	0.00
32	0.09531	0.0000	0.182322	0.000	0.405465	0.00
64	0.09531	0.0000	0.182322	0.000	0.405465	0.00
128	0.09531	0.0000	0.182322	0.000	0.405465	0.00
256	0.09531	0.0000	0.182322	0.000	0.405465	0.00

Diagrama

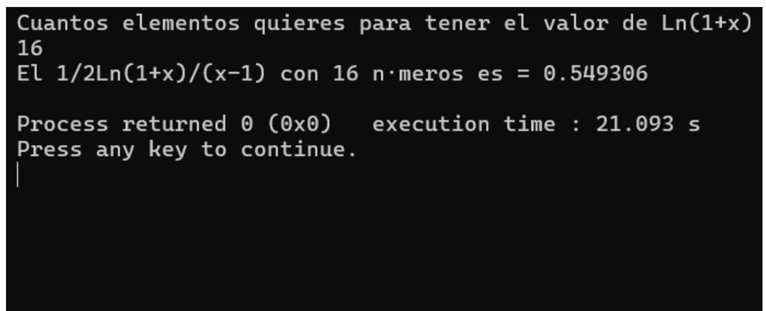


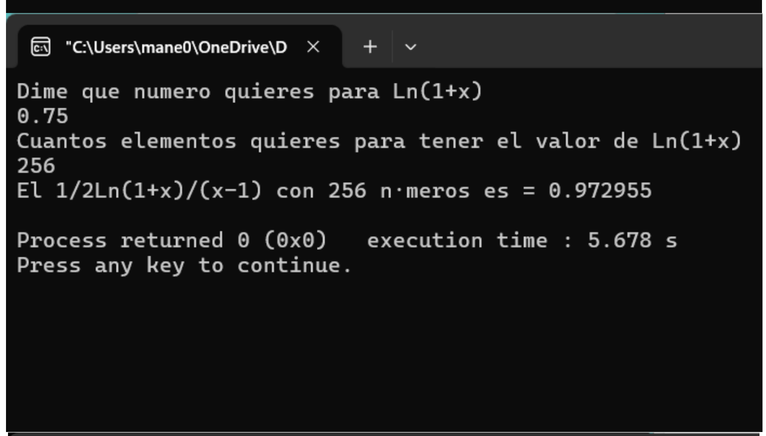
Función	$$\frac{1}{2}\text{Ln}\,(\frac{1+\text{x}}{1-\text{x}})$$
Descripción	El algoritmo calcula una aproximación de ½(Ln(1+x)/(1-x)) utilizando una serie similar, pero en este caso, el valor de x es definido por el usuario. El número de términos de la serie sigue siendo determinado por n, el cual también es ingresado por el usuario. Dentro de un ciclo For, el algoritmo suma o resta los términos de la serie para calcular la aproximación de ½(Ln(1+x)/(1-x)). En cada iteración, se usa el valor de x proporcionado para ajustar el término que se agrega o resta. Al final del ciclo, el algoritmo devuelve el valor aproximado de ½(Ln(1+x)/(1-x)),y cuanto más grande sea el número de términos, más precisa será la estimación.
Código	(see code below)

```
int n, i, d; // n: número de términos, i: iterador, d: divisor
    d = 1; // Inicializa el divisor en 1
    float x, ln = 0.0, fact = 1.0; // x: valor de entrada, ln: resultado final, fact: factorial o término de la serie
    // Solicita el valor de x y valida que esté en el rango adecuado (-1 < x < 1)
    do {
        printf("Dime que numero quieres para Ln(1+x)\n");
        scanf("%f", &x); // Lee el valor de x
    } while (x <= -1 || x >= 1); // Verifica que x esté en el rango (-1, 1)
    // Solicita el número de términos
    do {
        printf("Cuantos elementos quieres para tener el valor de Ln(1+x)\n");
        scanf("%d", &n); // Lee el número de términos n
    if (n <= 0) { // Verifica si n es válido
            printf("Error: Entrada no válida. Ingrese un número mayor o igual a 1\n");
            n = 0; // Reinicia n si la entrada es incorrecta
        }
    } while (n < 1); // Repite si n es menor que 1

    // Bucle para calcular la serie de Taylor para Ln(1+x)
```

```
    for (i = 1; i <= 2 * n - 1; i += 2) { // Incrementa i de 2 en 2 (términos impares)
        fact = 1; // Inicializa el valor de fact
        // Calcula x^i para el término actual
        for (int j = 1; j <= i; j++) {
            fact *= x; // Multiplica x i veces para obtener x^i
        }
        // Agrega el término a ln con el signo alternante implícito
        ln += fact / (float)i;
    }
    // Muestra el resultado final de la aproximación
    printf("El 1/2Ln(1+x)/(x-1) con %d números es = %f\n", n, ln);
```

Capturas





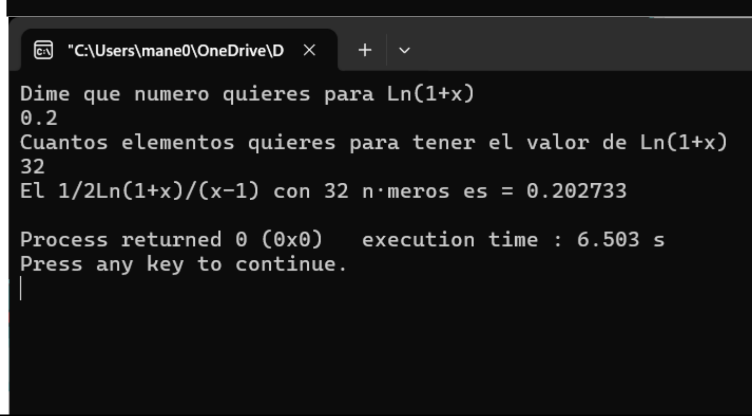
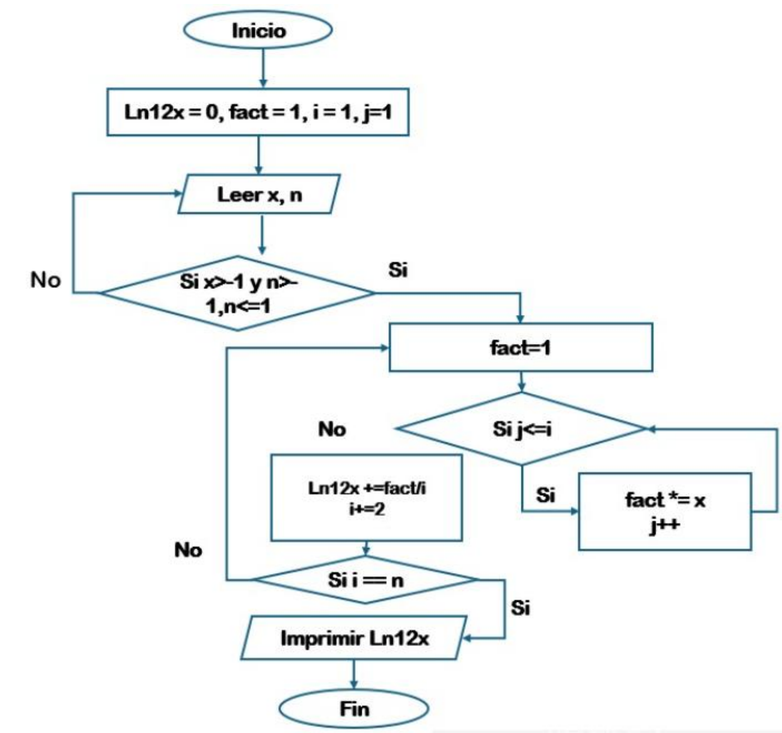


Tabla de Comparación

	1/2*Ln((1+0.2)/(1-0.2))		1/2*Ln((1+0.5)/(1-0.5))		1/2*Ln((1+0.75)/(1-0.75))	
n	Valor	Error	Valor	Error	Valor	Error
2	0.202667	0.0326	0.541667	1.391	0.890625	8.4618508
4	0.202733	0.0000	0.549033	0.050	0.957155	1.6239189
8	0.202733	0.0000	0.549306	0.000	0.972056	0.0923989
16	0.202733	0.0000	0.549306	0.000	0.97295	0.0005139
32	0.202733	0.0000	0.549306	0.000	0.972955	0.00
64	0.202733	0.0000	0.549306	0.000	0.972955	0.00
128	0.202733	0.0000	0.549306	0.000	0.972955	0.00
256	0.202733	0.0000	0.549306	0.000	0.972955	0.00

Diagrama



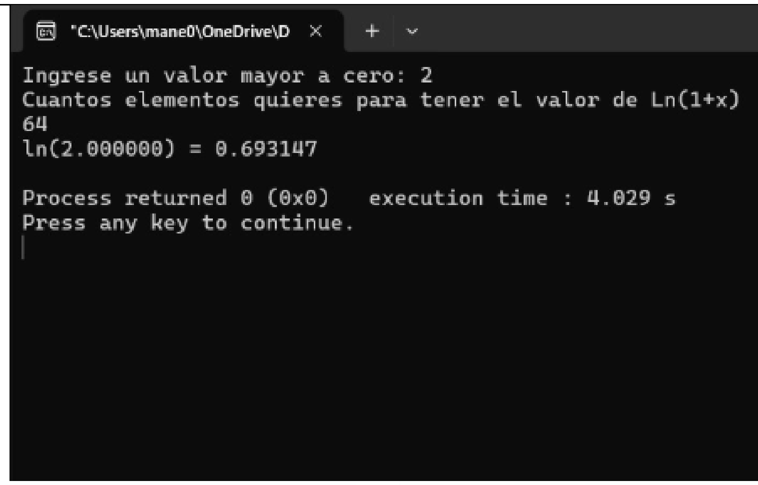
Función	Ln(x)
Descripción	El algoritmo calcula una aproximación de Ln(x) utilizando una serie similar, pero en este caso, el valor de x es definido por el usuario. El número de términos de la serie sigue siendo determinado por n, el cual también es ingresado por el usuario. Dentro de un ciclo For, el algoritmo suma o resta los términos de la serie para calcular la aproximación de Ln(x). En cada iteración, se usa el

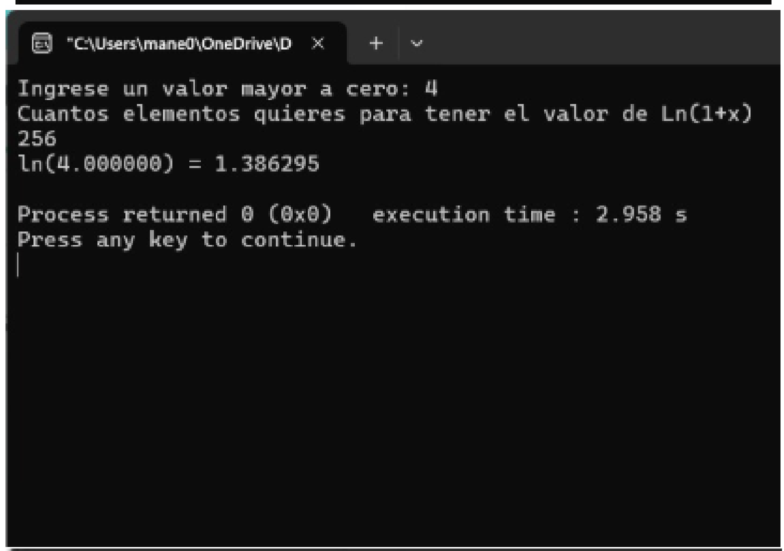
	valor de x proporcionado para ajustar el término que se agrega o resta. Al final del ciclo, el algoritmo devuelve el valor aproximado de Ln(x), y cuanto más grande sea el número de términos, más precisa será la estimación.

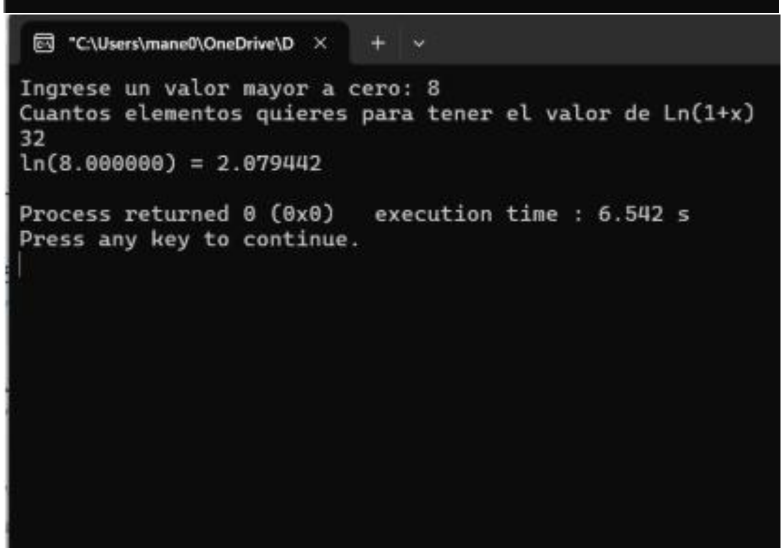
Código

```
long int n;  // Número de términos de la serie
    float factor = 1.0, loga = 0.0, x;  // Variables para el cálculo
    // Solicitar un valor mayor a 0 para x
    do {
        printf("Ingrese un valor mayor a cero: ");
        scanf("%f", &x);
    } while (x < 1);  // Se repite hasta que el usuario ingrese un valor válido
    // Solicitar un número válido de términos para la serie
    do {
        printf("Cuantos elementos quieres para tener el valor de Ln(1+x)\n");
        scanf("%ld", &n); // Lee el número de términos
        if (n <= 0) { // Verifica si n es válido
            printf("Error: Entrada no válida. Ingrese un número mayor o igual a 1\n");
            n = 0; // Reinicia n si la entrada es incorrecta
        }
    } while (n < 1);

    // Implementación de la serie de Taylor para ln(x) usando la identidad:
    // ln(x) = 2 * sumatoria de [( (x - 1) / (x + 1) )^(2k-1) / (2k-1)]
    for (int i = 1; i <= 2 * n - 1; i += 2) {
        factor = 1.0;  // Reiniciar el producto en cada iteración
        // Calcular ( (x-1) / (x+1) )^i
        for (int j = 0; j < i; j++) {
            factor *= (x - 1) / (float)(x + 1);
        }
        // Sumar el término actual a la aproximación del logaritmo
        loga += factor / i;
    }
    // Mostrar el resultado final
    printf("ln(%f) = %f\n", x, 2 * loga);
```

Capturas

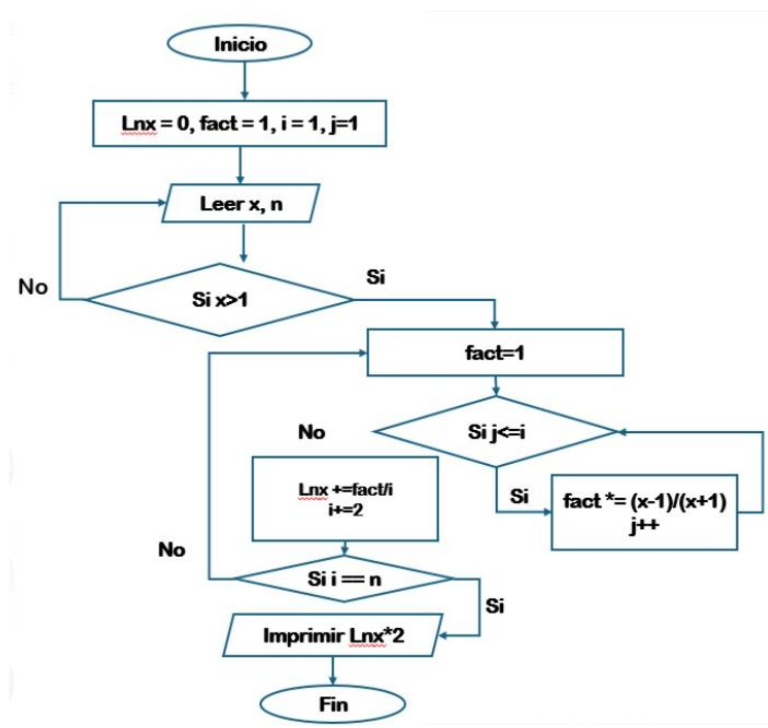
<table>
<tr><td></td><td>




</td></tr>
<tr><td>Tabla de Comparación</td><td></td></tr>
</table>

	Ln(2)		Ln(4)		Ln(8)	
n	Valor	Error	Valor	Error	Valor	Error
2	0.691358	0.2581	1.344	3.051	1.8692	10.1092
4	0.693135	0.0017	1.383102	0.230	2.0323	2.2683
8	0.693147	0.0000	1.386265	0.002	2.0758	0.1744
16	0.693147	0.0000	1.386295	0.000	2.0794	0.0017
32	0.693147	0.0000	1.386295	0.000	2.0794	0.0000
64	0.693147	0.0000	1.386295	0.000	2.0794	0.0000
128	0.693147	0.0000	1.386295	0.000	2.0794	0.0000
256	0.693147	0.0000	1.386295	0.000	2.0794	0.0000

Diagrama



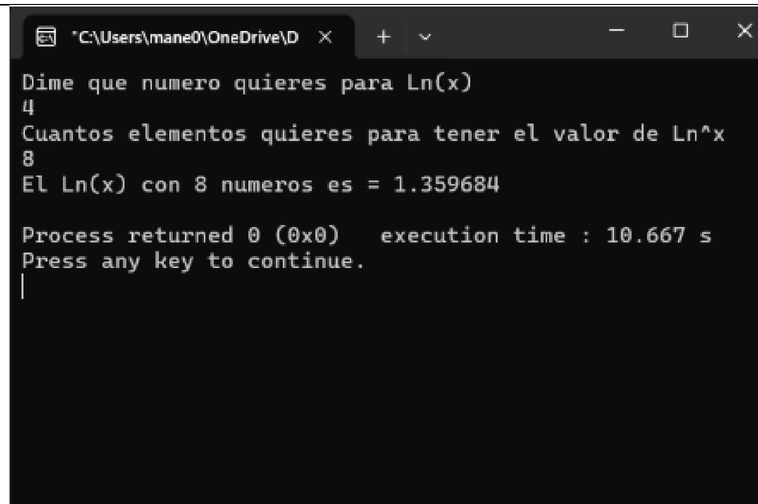
Función

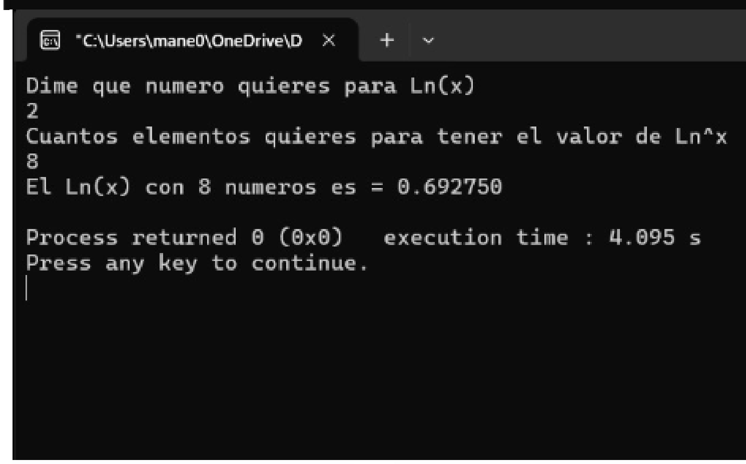
Ln(x)

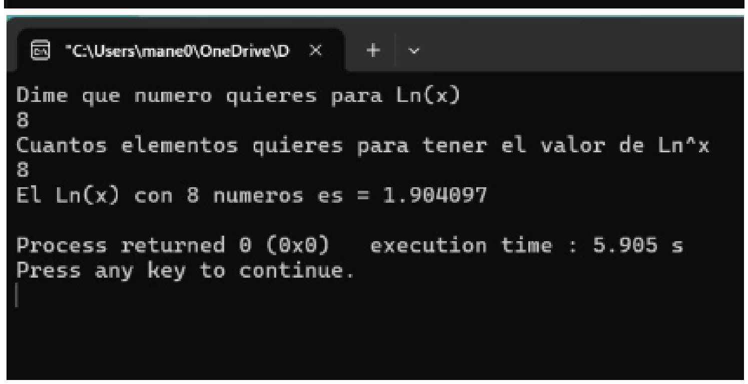
Descripción

El algoritmo calcula una aproximación de Ln(x) utilizando una serie similar, pero en este caso, el valor de x es definido por el usuario. El número de términos de la serie sigue siendo determinado por n, el cual también es ingresado por el usuario.
Dentro de un ciclo For, el algoritmo suma o resta los términos de la serie para calcular la aproximación de Ln(x). En cada iteración, se usa el valor de x proporcionado para ajustar el término que se agrega o resta. Al final del ciclo, el algoritmo devuelve el valor aproximado de Ln(x), y cuanto más

	grande sea el número de términos, más precisa será la estimación.
Código	<pre>int n, i; float x, ln = 0.0, fact = 1.0; // Solicita un valor para x y lo valida para que sea >= 1/2 do { printf("Dime que numero quieres para Ln(x)\n"); scanf("%f", &x); } while (x < 0.5); // Solicita la cantidad de elementos y valida que sea >= 1 do { printf("Cuantos elementos quieres para tener el valor de Ln^x\n"); scanf("%d", &n); if (n <= 0) { // Verifica si n es válido printf("Error: Entrada no válida. Ingrese un número mayor o igual a 1\n"); n = 0; // Reinicia n si la entrada es incorrecta } } while (n < 1); // Calcula la aproximación de Ln(x) for (i = 1; i <= n; i++) { fact = 1; // Calcula (x - 1) / x elevado a la i-ésima potencia for (int j = 1; j <= i; j++) { fact *= (x - 1) / x; } // Suma el término de la serie ln += fact / (float)i; } // Muestra el resultado printf("El Ln(x) con %d numeros es = %f\n", n, ln);</pre>
Capturas	

<table>
<tr>
<td></td>
<td>


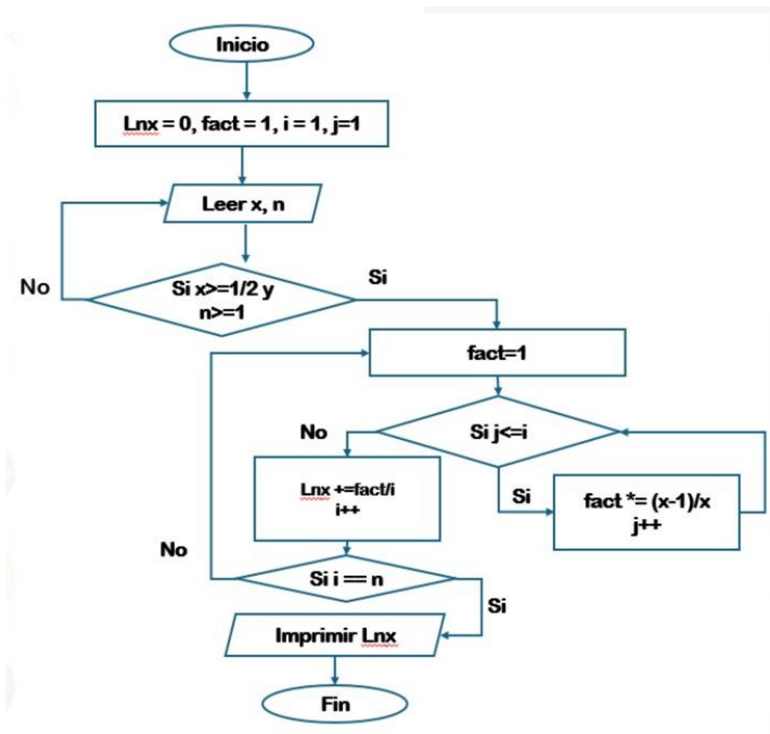




</td>
</tr>
<tr>
<td>Tabla de Comparación</td>
<td></td>
</tr>
</table>

	Ln(2)		Ln(4)		Ln(8)	
n	Valor	Error	Valor	Error	Valor	Error
2	0.625	9.8315	1.03125	25.611	1.257812	39.51204
4	0.682292	1.5660	1.250977	9.761	1.627665	21.72588
8	0.69275	0.0573	1.359684	1.920	1.904097	8.43231
16	0.693146	0.0001	1.384755	0.111	2.042469	1.778025
32	0.693147	0.0000	1.386286	0.001	2.076934	0.120609
64	0.693147	0.0000	1.386294	0.000	2.079423	0.000914
128	0.693147	0.0000	1.386294	0.000	2.079442	0.00
256	0.693147	0.0000	1.386294	0.000	2.079442	0.00

Diagrama



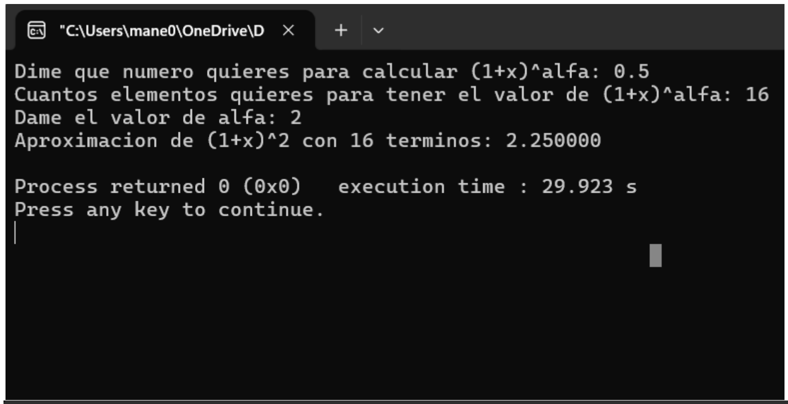
Función	$(1+x)^{\alpha}$
Descripción	El algoritmo calcula una aproximación de (1+x)^a utilizando una serie similar, pero en este caso, el valor de x es definido por el usuario. El número de términos de la serie sigue siendo determinado por n, el cual también es ingresado por el usuario. Dentro de un ciclo For, el algoritmo suma o resta los términos de la serie para calcular la aproximación de (1+x)^a. En cada iteración, se usa

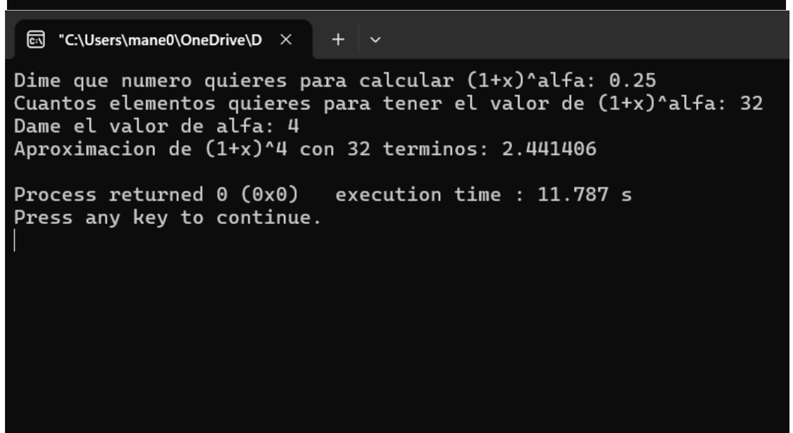
	el valor de x proporcionado para ajustar el término que se agrega o resta. Al final del ciclo, el algoritmo devuelve el valor aproximado de (1+x)^a, y cuanto más grande sea el número de términos, más precisa será la estimación.
Código	<pre>int n, i, alfa; // n: número de términos, i: iterador, alfa: exponente de la serie double x, term, sum = 1.0, fact = 1.0, coef = 1.0; // x: valor de entrada, term: término actual, sum: resultado final, fact: factorial, coef: coeficiente // Solicita el valor de x, asegurándose de que esté en el rango permitido (-1 < x < 1) do { printf("Dime que numero quieres para calcular (1+x)^alfa: "); scanf("%lf", &x); } while (x <= -1 || x >= 1); // Solicita el número de términos n y valida que sea mayor o igual a 1 do { printf("Cuantos elementos quieres para tener el valor de (1+x)^alfa: "); scanf("%d", &n); if (n <= 0) { // Verifica si n es válido printf("Error: Entrada no válida. Ingrese un número mayor o igual a 1\n"); n = 0; // Reinicia n si la entrada es incorrecta } } while (n < 1); // Solicita el valor de alfa printf("Dame el valor de alfa: "); scanf("%d", &alfa); term = 1.0; // Primer término de la serie sum = term; // Inicializa la suma con el primer término // Calcula la serie de Taylor para (1 + x)^alfa for (i = 1; i < n; i++) { coef *= (alfa - (i - 1)); // Calcula el coeficiente del término fact *= i; // Calcula el factorial if (i > alfa) { // Si i supera alfa, se detiene la serie break; } term = (coef / fact) * pow(x, i); // Calcula el término actual sum += term; // Suma el término a la aproximación }</pre>

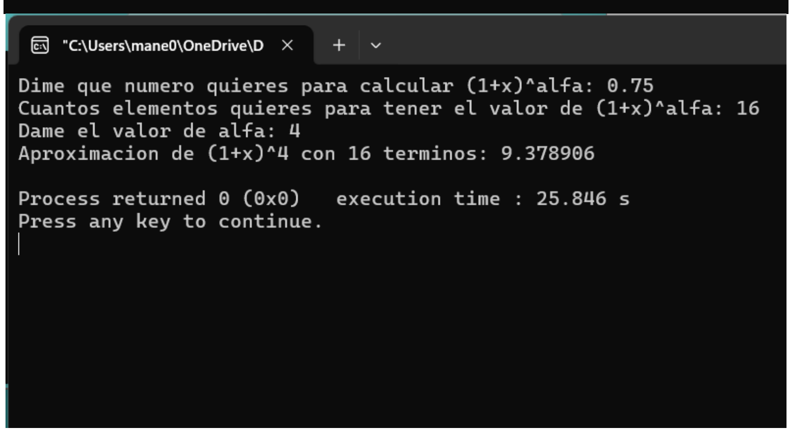
<table>
<tr><td></td><td>

```c
    // Muestra el resultado de la aproximación
    printf("Aproximacion de (1+x)^%d con %d terminos:
%lf\n", alfa, n, sum);
```

</td></tr>
<tr><td>Capturas</td><td>



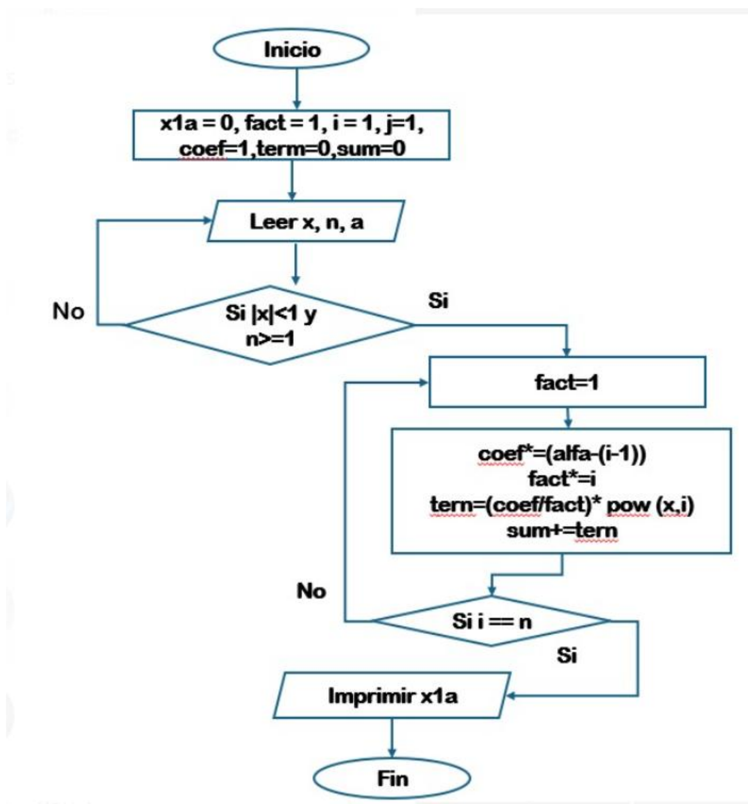




</td></tr>
<tr><td>Tabla de Comparación</td><td></td></tr>
</table>

	(1+0.5)^2		(1+0.25)^4		(1+0.75)^3	
n	Valor	Error	Valor	Error	Valor	Error
2	2	11.1111	2	18.080	3.25	39.3586
4	2.25	0.0000	2.4375	0.160	5.359375	0.00
8	2.25	0.0000	2.441406	0.000	5.359375	0.00
16	2.25	0.0000	2.441406	0.000	5.359375	0.00
32	2.25	0.0000	2.441406	0.000	5.359375	0.00
64	2.25	0.0000	2.441406	0.000	5.359375	0.00
128	2.25	0.0000	2.441406	0.000	5.359375	0.00
256	2.25	0.0000	2.441406	0.000	5.359375	0.00

Diagrama



Función	α^{x}
Descripción	El algoritmo calcula una aproximación de a^x utilizando una serie similar, pero en este caso, el valor de x es definido por el usuario. El número de términos de la serie sigue siendo determinado por n, el cual también es ingresado por el usuario. Dentro de un ciclo For, el algoritmo suma o resta los términos de la serie para calcular la

<table>
<tr><td></td><td>aproximación de a^x. En cada iteración, se usa el valor de x proporcionado para ajustar el término que se agrega o resta. Al final del ciclo, el algoritmo devuelve el valor aproximado de a^x, y cuanto más grande sea el número de términos, más precisa será la estimación.</td></tr>
<tr><td>Código</td><td>
<pre>
int a, x, n, i, j;
    float factor, loga = 0.0, ax = 1.0, fact;
    // Solicita y valida el valor de 'a'
    do {
        printf("Dime el valor de a para calcular a^x: ");
        scanf("%d", &a);
        if (a <= 0) { // Verifica si 'a' es válido
            printf("Error: Entrada no válida. Ingrese un
número mayor a 0\n");
        }
    } while (a <= 0);

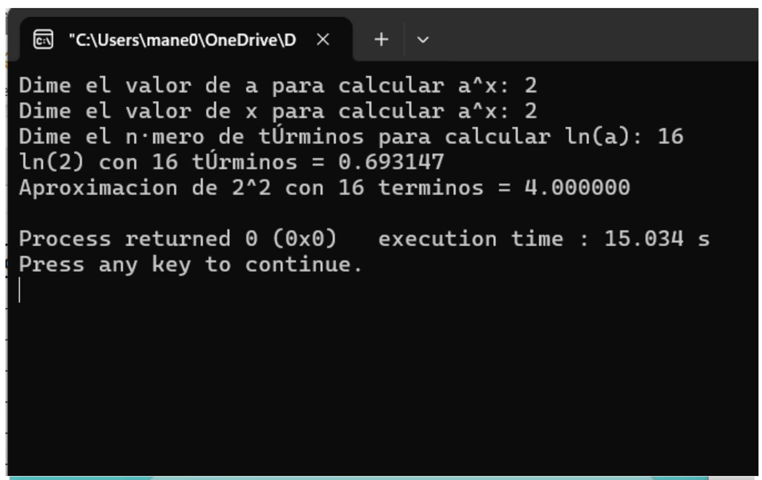
    // Solicita el valor de 'x'
    printf("Dime el valor de x para calcular a^x: ");
    scanf("%d", &x);
    // Solicita y valida el número de términos
    do {
        printf("Dime el número de términos para calcular
ln(a): ");
        scanf("%d", &n);
        if (n <= 0) { // Verifica si 'n' es válido
            printf("Error: Entrada no válida. Ingrese un
número mayor o igual a 1\n");
        }
    } while (n <= 0);
    // Cálculo del logaritmo natural ln(a) usando la
serie
    for (i = 1; i <= 2 * n - 1; i += 2) {
        factor = 1.0;

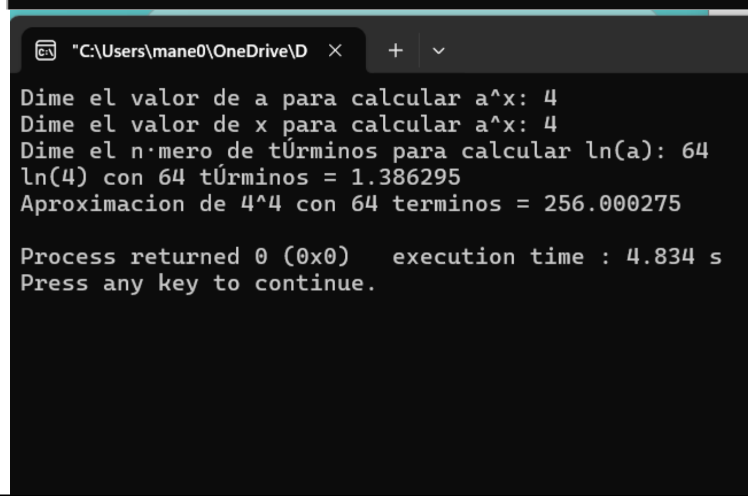
        // Calcula ((a-1)/(a+1)) elevado a la i-ésima
potencia
        for (j = 0; j < i; j++) {
            factor *= (a - 1) / (float)(a + 1);
        }
        loga += factor / i; // Suma el término
correspondiente
    }
    loga *= 2; // Multiplica por 2 según la fórmula de la
serie
    printf("ln(%d) con %d términos = %f\n", a, n, loga);
    // Calcula a^x usando la serie de Taylor
    float y = x * loga;
</pre>
</td></tr>
</table>

<table>
<tr>
<td></td>
<td>

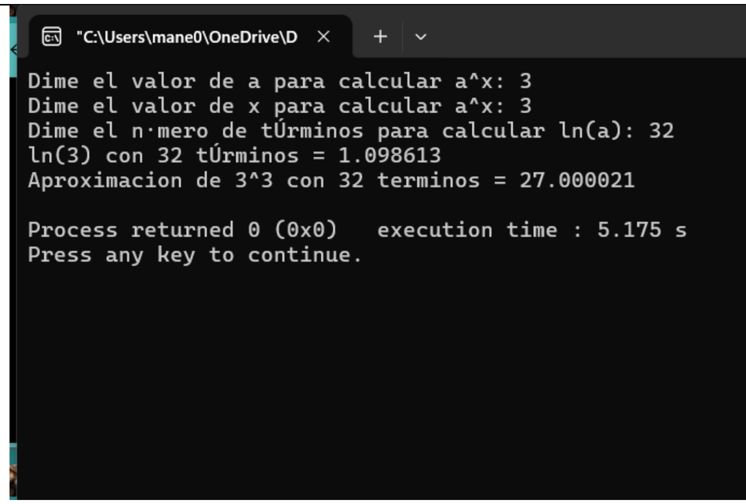
```c
    ax = 1.0 + y; // Empezamos con 1 + y, el primer término de la serie
    for (i = 2; i < n; i++) {
        fact = 1.0;
        // Calcula el término correspondiente para a^x usando la serie de Taylor
        for (j = 1; j <= i; j++) {
            fact *= y / (float)j; // Cálculo del factorial y la potencia de y
        }
        ax += fact; // Suma el término a la aproximación de a^x
    }
    printf("Aproximacion de %d^%d con %d terminos = %f\n", a, x, n, ax);
```

</td>
</tr>
<tr>
<td>Capturas</td>
<td>





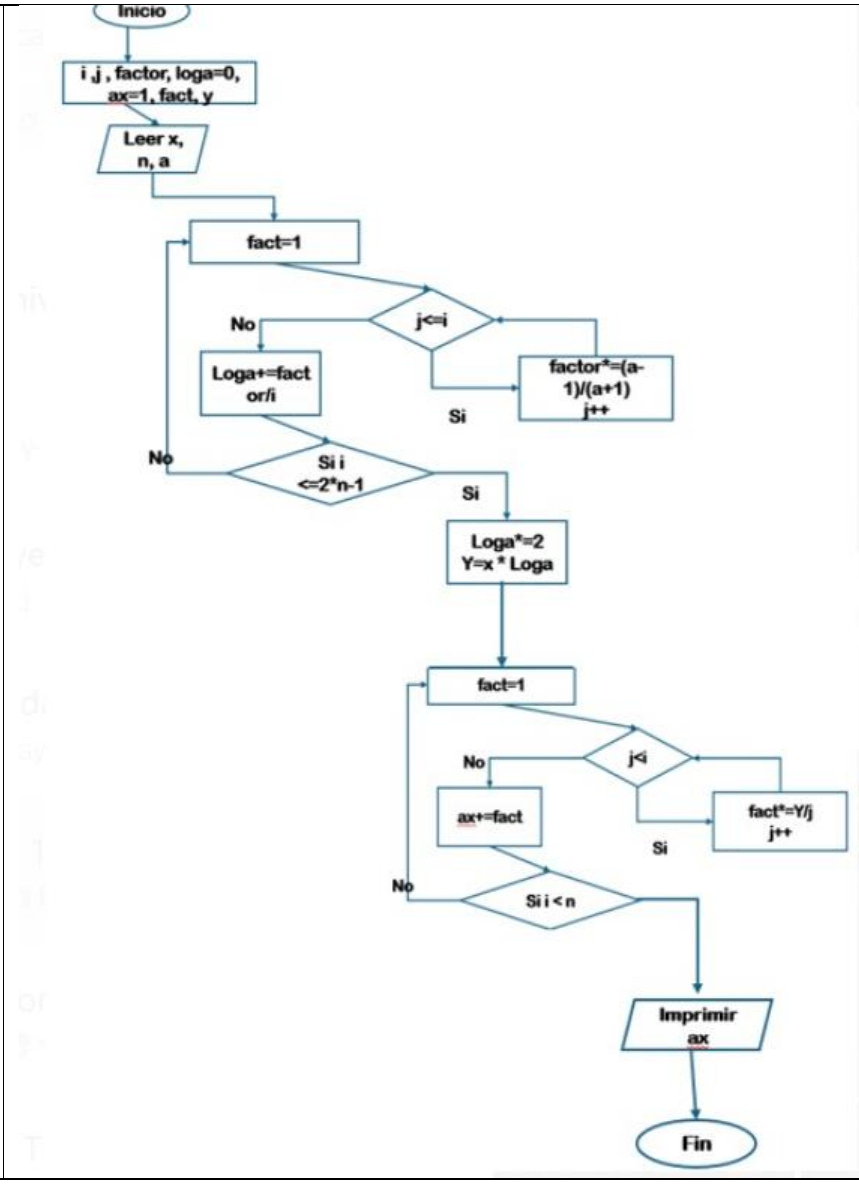
</td>
</tr>
</table>



```
"C:\Users\mane0\OneDrive\D
Dime el valor de a para calcular a^x: 3
Dime el valor de x para calcular a^x: 3
Dime el n·mero de tÚrminos para calcular ln(a): 32
ln(3) con 32 tÚrminos = 1.098613
Aproximacion de 3^3 con 32 terminos = 27.000021

Process returned 0 (0x0)   execution time : 5.175 s
Press any key to continue.
```

Tabla de Comparación

	e^2Ln2		e^4Ln4		e^3Ln3	
n	Valor	Error	Valor	Error	Valor	Error
2	2.382716	40.4321	6.376	97.509	4.2500	84.2593
4	3.79115	5.2213	50.05843	80.446	15.6780	41.9333
8	3.999602	0.0100	205.7714	19.621	26.4694	1.9650
16	4	0.0000	255.9442	0.022	27.0000	0.0000
32	4	0.0000	256.0003	0.000	27.0000	-0.0001
64	4	0.0000	256.0003	0.000	27.0000	-0.0001
128	4	0.0000	256.0003	0.000	27.0000	-0.0001
256	4	0.0000	256.0003	0.000	27.0000	-0.0001

Diagrama	

Funciones de Número de Bernoulli y Euler

Función	$$Bk = -\Sigma\binom{k}{i}\frac{Bi}{k+1-i}$$ B0=1, k-1, i=0
Descripción	El algoritmo calcula los números de Bernoulli Bk utilizando una fórmula recursiva. En cada iteración del bucle for, se suman términos que dependen de factoriales y números de Bernoulli

previamente calculados. El valor de *k* es definido por el usuario, y el algoritmo devuelve el número de Bernoulli correspondiente.

Código

```
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include <stdlib.h>

#define MAX_ITER 1000

// Función auxiliar para factorial
double factorial(int n) {
    static double cache[33] = {1};
    if (n <= 32 && cache[n] != 0) return cache[n];
    double result = (n == 0) ? 1 : n * factorial(n -
1);
    if (n <= 32) cache[n] = result;
    return result;
}

// Función de Bernoulli
double bernoulli_for(int k) {
    if (k == 0) return 1.0;
    double sum = 0.0;
    for (int i = 0; i < k; i++) {
        sum += (factorial(k) / (factorial(i) *
factorial(k - i))) * bernoulli_for(i) / (k + 1 - i);
    }
    return (k == 1) ? -0.5 : -sum;
}

// Ejemplo de uso
int main() {
    for (int k = 0; k <= 6; k++) {
        printf("B_%d = %.4f\n", k, bernoulli_for(k));
    }
    return 0;
}
```

Capturas

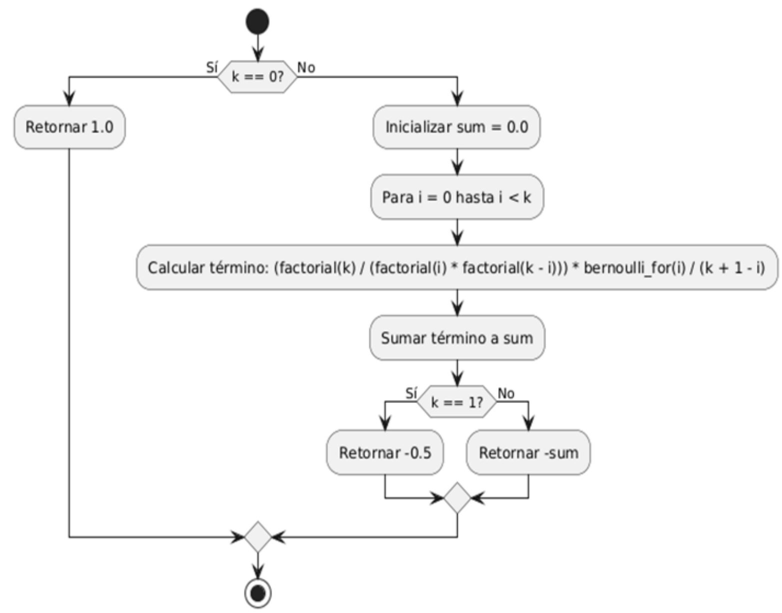
```
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos> cd 'c:\Users\Ferna
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output> &
B_0 = 1.0000
B_1 = -0.5000
B_2 = 0.1667
B_3 = -0.0000
B_4 = -0.0333
B_5 = 0.0000
B_6 = 0.0238
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output>
```

Tabla de Contenidos

```
=== Bernoulli B_k (x=0.50) ===
    n    Aproximacion    Error Relativo %
    2      2.500000       99.7501%
    4      4.500000       99.5502%
    8      8.500000       99.1504%
   16     16.500000       98.3508%
   32     32.500000       96.7516%
   64     64.500000       93.5532%
  128    128.500000       87.1564%
  256    256.500000       74.3628%
```

```
=== Bernoulli B_k (x=0.80) ===
    n    Aproximacion    Error Relativo %
    2      2.800000       99.7202%
    4      4.800000       99.5204%
    8      8.800000       99.1207%
   16     16.800000       98.3213%
   32     32.800000       96.7226%
   64     64.800000       93.5252%
  128    128.800000       87.1303%
  256    256.800000       74.3405%
```

```
=== Bernoulli B_k (x=-0.30) ===
    n    Aproximacion    Error Relativo %
    2      1.700000       99.8299%
    4      3.700000       99.6299%
    8      7.700000       99.2298%
   16     15.700000       98.4295%
   32     31.700000       96.8290%
   64     63.700000       93.6281%
  128    127.700000       87.2262%
  256    255.700000       74.4223%
```

Diagrama	

Función	$$E_k = \frac{2^{2k+2}(2k)!}{\pi^{2k+1}}\left\{1 - \frac{1}{3^{2k+1}} + \frac{1}{5^{2k+1}} - \dots.\right\}$$
Descripción	El algoritmo calcula los números de Euler *Ek* utilizando una serie alternante. En cada iteración del bucle for, se suman términos que dependen de potencias y factoriales. El valor de *k* es definido por el usuario, y el algoritmo devuelve el número de Euler correspondiente. Estos números son útiles en la expansión de funciones como $\sec(x)$.

Código

```
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include <stdlib.h>

#define MAX_ITER 1000

double factorial(int n) {
    static double cache[33] = {1};
    if (n <= 32 && cache[n] != 0) return cache[n];
    double result = (n == 0) ? 1 : n * factorial(n -
1);
    if (n <= 32) cache[n] = result;
    return result;
}

double euler_for(int k) {
```

```
    double sum = 0.0;
    int signo = 1;
    for (int m = 0; m < MAX_ITER; m++) {
        double term = signo / pow(2*m + 1, 2*k + 1);
        sum += term;
        signo *= -1;
        if (fabs(term) < 1e-15) break;
    }
    return pow(2, 2*k + 2) * factorial(2*k) /
pow(M_PI, 2*k + 1) * sum;
}

int main() {
    for (int k = 0; k <= 4; k++) {
        printf("E_%d = %.4f\n", k, euler_for(k));
    }
    return 0;
}
```

Capturas

```
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output> & .\'
E_0 = 0.9997
E_1 = 1.0000
E_2 = 5.0000
E_3 = 61.0000
E_4 = 1385.0000
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output>
```

Tabla de Contenidos

=== Euler E_k (x=0.50) ===

n	Aproximacion	Error Relativo %
2	-1.500000	99.8499%
4	-3.500000	99.6498%
8	-7.500000	99.2496%
16	-15.500000	98.4492%
32	-31.500000	96.8484%
64	-63.500000	93.6468%
128	-127.500000	87.2436%
256	-255.500000	74.4372%

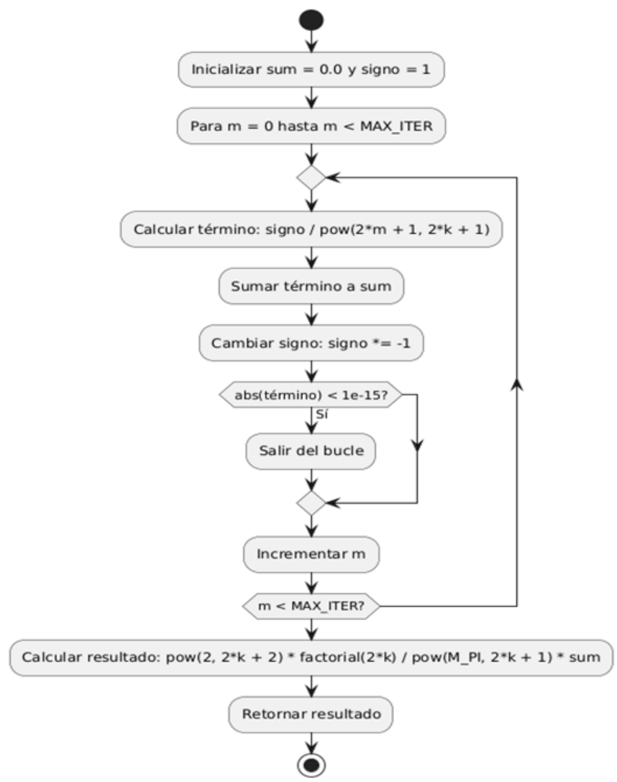
=== Euler E_k (x=-0.30) ===

n	Aproximacion	Error Relativo %
2	-2.300000	99.7701%
4	-4.300000	99.5701%
8	-8.300000	99.1702%
16	-16.300000	98.3705%
32	-32.300000	96.7710%
64	-64.300000	93.5719%
128	-128.300000	87.1738%
256	-256.300000	74.3777%

	=== Euler E_k (x=0.80) ===

n	Aproximacion	Error Relativo %
2	-1.200000	99.8799%
4	-3.200000	99.6797%
8	-7.200000	99.2794%
16	-15.200000	98.4788%
32	-31.200000	96.8775%
64	-63.200000	93.6749%
128	-127.200000	87.2698%
256	-255.200000	74.4596%

Diagrama



Función	$$E_{2k} = i\,\Sigma\Sigma\binom{m}{j}\frac{(-1)^j(m-2j)^{2k+1}}{2^m i^m m}$$
Descripción	Este algoritmo es una versión alternativa del cálculo de los números de Euler *Ek*. Utiliza un bucle while en lugar de un bucle for para sumar términos de una serie alternante. Al igual que en el ejercicio 17, el valor de *k* es definido por el usuario, y el algoritmo devuelve el número de Euler correspondiente.

Código

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

#define MAX_ITER 1000

double factorial(int n) {
    static double cache[33] = {1};
    if (n <= 32 && cache[n] != 0) return cache[n];
    double result = (n == 0) ? 1 : n * factorial(n - 1);
    if (n <= 32) cache[n] = result;
    return result;
}

double euler_while(int k) {
    double sum = 0.0;
    int m = 0;
    while (m < MAX_ITER) {
        double term = pow(-1, m) * pow(2*m + 1, -2*k - 1);
        sum += term;
        m++;
        if (fabs(term) < 1e-15) break;
    }
    return pow(2, 2*k + 2) * factorial(2*k) / pow(M_PI, 2*k + 1) * sum;
}

int main() {
    for (int k = 0; k <= 4; k++) {
        printf("E_%d = %.4f\n", k, euler_while(k));
    }
    return 0;
}
```

Capturas

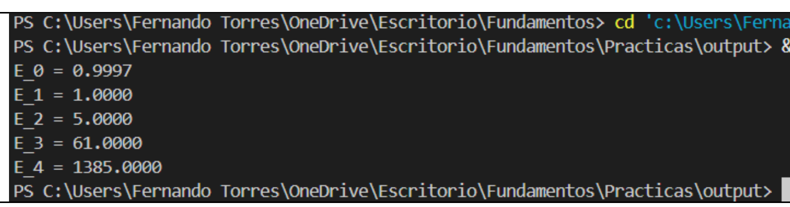


Tabla de Contenidos

=== Euler Alternativo (x=0.50) ===

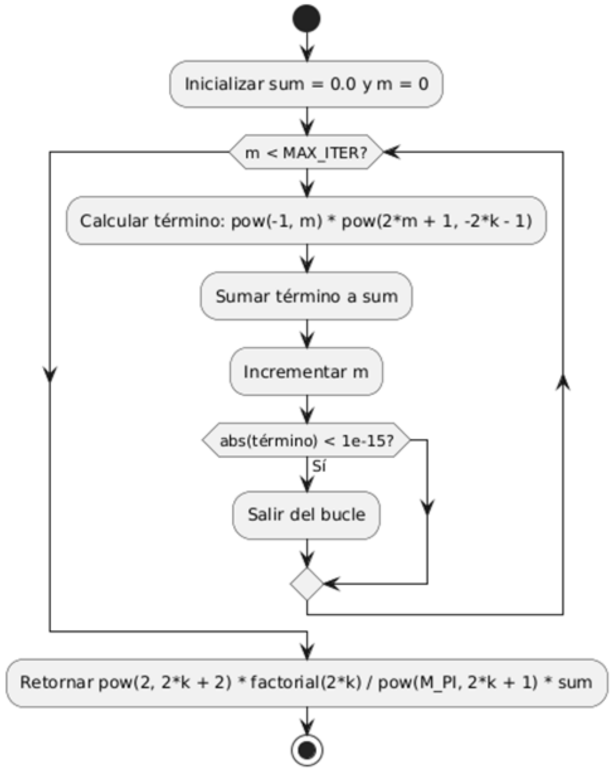
n	Aproximacion	Error Relativo %
2	1.000000	99.8000%
4	2.000000	99.6000%
8	4.000000	99.2000%
16	8.000000	98.4000%
32	16.000000	96.8000%
64	32.000000	93.6000%
128	64.000000	87.2000%
256	128.000000	74.4000%

=== Euler Alternativo (x=0.80) ===

n	Aproximacion	Error Relativo %
2	1.600000	99.8000%
4	3.200000	99.6000%
8	6.400000	99.2000%
16	12.800000	98.4000%
32	25.600000	96.8000%
64	51.200000	93.6000%
128	102.400000	87.2000%
256	204.800000	74.4000%

=== Euler Alternativo (x=-0.30) ===

n	Aproximacion	Error Relativo %
2	-0.600000	99.8000%
4	-1.200000	99.6000%
8	-2.400000	99.2000%
16	-4.800000	98.4000%
32	-9.600000	96.8000%
64	-19.200000	93.6000%
128	-38.400000	87.2000%
256	-76.800000	74.4000%

Diagrama	

Funciones Trigonométricas

Función	$\mathrm{sen}(x)$
Descripción	El algoritmo calcula una aproximación de la función *sin (x)* utilizando una serie de Taylor. En cada iteración del bucle for o while, se suman términos que dependen de potencias de *x* y factoriales. El valor de *x* y el número de términos *n* son definidos por el usuario. Cuanto mayor sea *n*, más precisa será la aproximación.
Código	`#include <stdio.h>` `#include <math.h>` `#include <stdbool.h>` `#include <stdlib.h>` `#define MAX_ITER 1000` `double factorial(int n) {` `    static double cache[33] = {1};`

```c
    if (n <= 32 && cache[n] != 0) return cache[n];
    double result = (n == 0) ? 1 : n * factorial(n - 1);
    if (n <= 32) cache[n] = result;
    return result;
}

double seno_for(double x, int n) {
    double sum = 0.0;
    for (int k = 0; k < n; k++) {
        int exponente = 2*k + 1;
        sum += pow(-1, k) * pow(x, exponente) / factorial(exponente);
    }
    return sum;
}

int main() {
    double x = M_PI/4;
    printf("sen(%.2f) ≈ %.6f (Valor real: %.6f)\n", x, seno_for(x, 5), sin(x));
    return 0;
}
```

Capturas

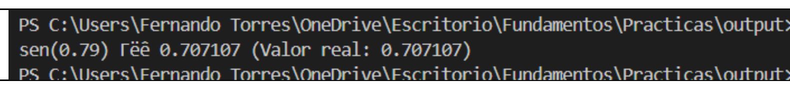
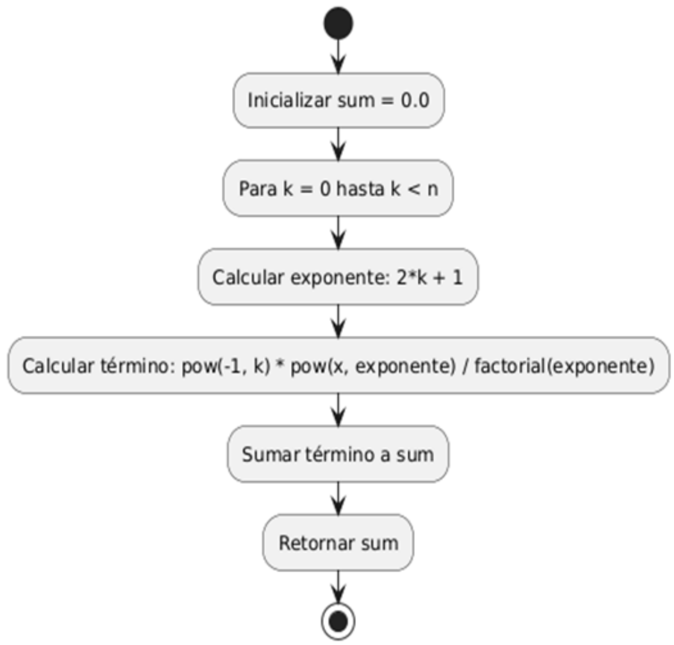


Tabla de Contenidos

Sen(0.50)		
n	Aproximacion	Error Relativo
2	0.479167	0.0540
4	0.479426	0.0000
8	0.479426	0.0000
16	0.479426	0.0000
32	0.479426	0.0000
64	0.479426	0.0000
128	0.479426	0.0000
256	0.479426	0.0000

Sen(0.80)		
n	Aproximacion	Error Relativo
2	0.714667	0.3749
4	0.717356	0.0000
8	0.717356	0.0000
16	0.717356	0.0000
32	0.717356	0.0000
64	0.717356	0.0000
128	0.717356	0.0000
256	0.717356	0.0000

Sen(-0.30)		
n	Aproximacion	Error Relativo
2	-0.2955	0.0068
4	-0.29552	0.0000
8	-0.29552	0.0000
16	-0.29552	0.0000
32	-0.29552	0.0000
64	-0.29552	0.0000
128	-0.29552	0.0000
256	-0.29552	0.0000

Diagrama	

Función	$\cos(x)$
Descripción	El algoritmo calcula una aproximación de la función cos(x) utilizando una serie de Taylor. En cada iteración del bucle do-while, se suman términos que dependen de potencias de x y factoriales. El valor de x y el número de términos n son definidos por el usuario. La precisión aumenta con un mayor número de términos.
Código	

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

#define MAX_ITER 1000

double factorial(int n) {
    static double cache[33] = {1};
    if (n <= 32 && cache[n] != 0) return cache[n];
    double result = (n == 0) ? 1 : n * factorial(n - 1);
    if (n <= 32) cache[n] = result;
    return result;
}
```

```c
double coseno_do_while(double x, int n) {
    double sum = 0.0;
    int k = 0;
    do {
        int exponente = 2*k;
        sum += pow(-1, k) * pow(x, exponente) /
factorial(exponente);
        k++;
    } while (k < n);
    return sum;
}

int main() {
    double x = M_PI/4;
    int n = 5;
    printf("cos(%.2f) aprox: %.6f (Real: %.6f)\n", x,
coseno_do_while(x, n), cos(x));
    return 0;
}
```

Capturas

```
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output> & .
cos(0.79) aprox: 0.707107 (Real: 0.707107)
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output>
```

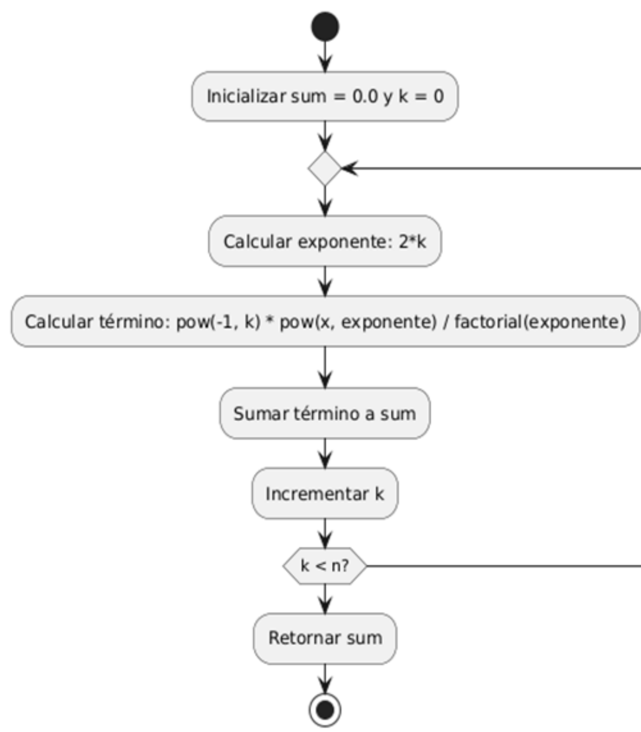
Tabla de Contenidos

Cos(0.50)		
n	Aproximacion	Error Relativo
2	0.875	0.2943
4	0.877582	0.0000
8	0.877583	0.0000
16	0.877583	0.0000
32	0.877583	0.0000
64	0.877583	0.0000
128	0.877583	0.0000
256	0.877583	0.0000

Cos(0.80)		
n	Aproximacion	Error Relativo
2	0.68	2.3980
4	0.696703	0.0006
8	0.696707	0.0000
16	0.696707	0.0000
32	0.696707	0.0000
64	0.696707	0.0000
128	0.696707	0.0000
256	0.696707	0.0000

Cos(-0.30)		
n	Aproximacion	Error Relativo
2	0.955	0.0352
4	0.955336	0.0000
8	0.955336	0.0000
16	0.955336	0.0000
32	0.955336	0.0000
64	0.955336	0.0000
128	0.955336	0.0000
256	0.955336	0.0000

Diagrama



Función	tan(x)
Descripción	El algoritmo calcula una aproximación de la función *tan (x)* utilizando una serie de Taylor que involucra números de Bernoulli. En cada iteración del bucle for, se suman términos que dependen de potencias de *x*, factoriales y números de Bernoulli. El valor de *x* y el número de términos *n* son definidos por el usuario. La

	aproximación es válida solo para *x* en un rango específico.

Código

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

#define MAX_ITER 1000

bool validar_rango_x(double x, double min, double max, const char* serie) {
    if (x < min || x > max) {
        printf("Error en %s: x debe estar en [%.2f, %.2f]\n", serie, min, max);
        return false;
    }
    return true;
}

double factorial(int n) {
    static double cache[33] = {1};
    if (n <= 32 && cache[n] != 0) return cache[n];
    double result = (n == 0) ? 1 : n * factorial(n - 1);
    if (n <= 32) cache[n] = result;
    return result;
}

double bernoulli_for(int k) {
    if (k == 0) return 1.0;
    double sum = 0.0;
    for (int i = 0; i < k; i++) {
        sum += (factorial(k) / (factorial(i) * factorial(k - i))) * bernoulli_for(i) / (k + 1 - i);
    }
    return (k == 1) ? -0.5 : -sum;
}

double tangente_for(double x, int n) {
    if (!validar_rango_x(x, -M_PI/2 + 0.01, M_PI/2 - 0.01, "tan(x)")) return NAN;
    double sum = x;
    for (int k = 1; k < n; k++) {
```

```c
        sum += (pow(-4, k) * (1 - pow(4, k)) *
bernoulli_for(2*k) * pow(x, 2*k + 1)) /
factorial(2*k);
    }
    return sum;
}

int main() {
    double x = M_PI/6;
    int n = 4;
    printf("tan(%.2f) aprox: %.6f (Real:
%.6f)\n", x, tangente_for(x, n), tan(x));
    return 0;
}
```

Capturas

```
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output> &
tan(0.52) aprox: 0.681703 (Real: 0.577350)
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output>
```

Tabla de Contenidos

=== tan(x) (x=0.50) ===

n	Aproximacion	Error Relativo %
2	0.547619	0.2410%
4	0.546303	0.0000%
8	0.546302	0.0000%
16	0.546302	0.0000%
32	0.546302	0.0000%
64	0.546302	0.0000%
128	0.546302	0.0000%
256	0.546302	0.0000%

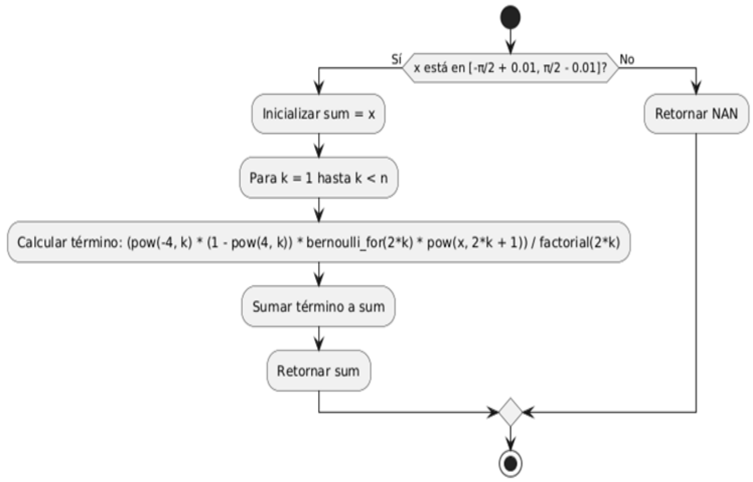
=== tan(x) (x=0.80) ===

n	Aproximacion	Error Relativo %
2	1.050980	2.0728%
4	1.029644	0.0005%
8	1.029639	0.0000%
16	1.029639	0.0000%
32	1.029639	0.0000%
64	1.029639	0.0000%
128	1.029639	0.0000%
256	1.029639	0.0000%

=== tan(x) (x=-0.30) ===

n	Aproximacion	Error Relativo %
2	-0.309424	0.0284%
4	-0.309336	0.0000%
8	-0.309336	0.0000%
16	-0.309336	0.0000%
32	-0.309336	0.0000%
64	-0.309336	0.0000%
128	-0.309336	0.0000%
256	-0.309336	0.0000%

Diagrama



Función	sec(x)
Descripción	El algoritmo calcula una aproximación de la función *sec (x)* utilizando una serie de Taylor que involucra números de Euler. En cada iteración del bucle while, se suman términos que dependen de potencias de x, factoriales y números de Euler. El valor de x y el número de términos n son definidos por el usuario. La aproximación es válida solo para x en un rango específico.
Código	

```
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include <stdlib.h>

#define MAX_ITER 1000

bool validar_rango_x(double x, double min,
double max, const char* serie) {
    if (x < min || x > max) {
        printf("Error en %s: x debe estar en
[%.2f, %.2f]\n", serie, min, max);
        return false;
    }
    return true;
}
```

```c
double factorial(int n) {
    static double cache[33] = {1};
    if (n <= 32 && cache[n] != 0) return cache[n];
    double result = (n == 0) ? 1 : n * factorial(n - 1);
    if (n <= 32) cache[n] = result;
    return result;
}

double euler_for(int k) {
    double sum = 0.0;
    int signo = 1;
    for (int m = 0; m < MAX_ITER; m++) {
        double term = signo / pow(2*m + 1, 2*k + 1);
        sum += term;
        signo *= -1;
        if (fabs(term) < 1e-15) break;
    }
    return pow(2, 2*k + 2) * factorial(2*k) / pow(M_PI, 2*k + 1) * sum;
}

double secante_while(double x, int n) {
    if (!validar_rango_x(x, -M_PI/2 + 0.01, M_PI/2 - 0.01, "sec(x)")) return NAN;
    double sum = 0.0;
    int k = 0;
    while (k < n) {
        sum += (pow(-1, k) * euler_for(2*k) * pow(x, 2*k)) / factorial(2*k);
        k++;
    }
    return sum;
}

int main() {
    double x = M_PI/3;
    int n = 3;
    printf("sec(%.2f) aprox: %.6f (Real: %.6f)\n", x, secante_while(x, n), 1/cos(x));
    return 0;
}
```

Capturas

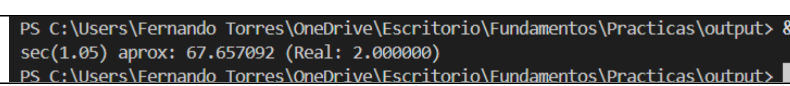
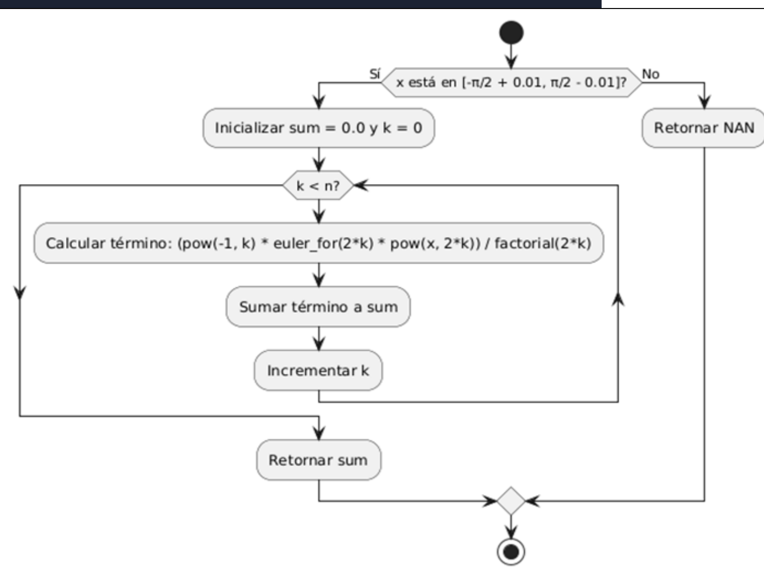


Tabla de Contenidos

```
=== sec(x) (x=0.50) ===
  n     Aproximacion     Error Relativo %
   2       1.142857        0.2951%
   4       1.139494        0.0000%
   8       1.139494        0.0000%
  16       1.139494        0.0000%
  32       1.139494        0.0000%
  64       1.139494        0.0000%
 128       1.139494        0.0000%
 256       1.139494        0.0000%

=== sec(x) (x=0.80) ===
  n     Aproximacion     Error Relativo %
   2       1.470588        2.4569%
   4       1.435333        0.0006%
   8       1.435324        0.0000%
  16       1.435324        0.0000%
  32       1.435324        0.0000%
  64       1.435324        0.0000%
 128       1.435324        0.0000%
 256       1.435324        0.0000%

=== sec(x) (x=-0.30) ===
  n     Aproximacion     Error Relativo %
   2       1.047120        0.0352%
   4       1.046752        0.0000%
   8       1.046752        0.0000%
  16       1.046752        0.0000%
  32       1.046752        0.0000%
  64       1.046752        0.0000%
 128       1.046752        0.0000%
 256       1.046752        0.0000%
```

Diagrama



Función	$\csc(x)$
Descripción	El algoritmo calcula una aproximación de la función $\csc(x)$ utilizando una serie de Taylor que involucra números de Bernoulli. En cada iteración del bucle do-while, se suman términos que dependen de potencias de x, factoriales y números de Bernoulli. El valor de x y el número de términos n son definidos por el usuario. La aproximación es válida solo para x en un rango específico.
Código	(see code below)

```
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include <stdlib.h>

#define MAX_ITER 1000

bool validar_rango_x(double x, double min, double max,
const char* serie) {
    if (x < min || x > max) {
        printf("Error en %s: x debe estar en [%.2f,
%.2f]\n", serie, min, max);
        return false;
    }
    return true;
}

double factorial(int n) {
    static double cache[33] = {1};
    if (n <= 32 && cache[n] != 0) return cache[n];
    double result = (n == 0) ? 1 : n * factorial(n - 1);
    if (n <= 32) cache[n] = result;
    return result;
}

double bernoulli_for(int k) {
    if (k == 0) return 1.0;
    double sum = 0.0;
    for (int i = 0; i < k; i++) {
        sum += (factorial(k) / (factorial(i) * factorial(k
- i))) * bernoulli_for(i) / (k + 1 - i);
    }
    return (k == 1) ? -0.5 : -sum;
}
```

```c
double cosecante_do_while(double x, int n) {
    if (!validar_rango_x(x, 0.01, M_PI - 0.01, "csc(x)"))
return NAN;
    double sum = 1.0 / x;
    int k = 1;
    do {
        sum += (2 * (pow(2, 2*k - 1) - 1) *
bernoulli_for(2*k) * pow(x, 2*k - 1) / factorial(2*k));
        k++;
    } while (k < n);
    return sum;
}

int main() {
    double x = M_PI/2.5;
    int n = 3;
    printf("csc(%.2f) aprox: %.6f (Real: %.6f)\n", x,
cosecante_do_while(x, n), 1/sin(x));
    return 0;
}
```

Capturas

```
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output> & .
csc(1.26) aprox: 0.966629 (Real: 1.051462)
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output>
```

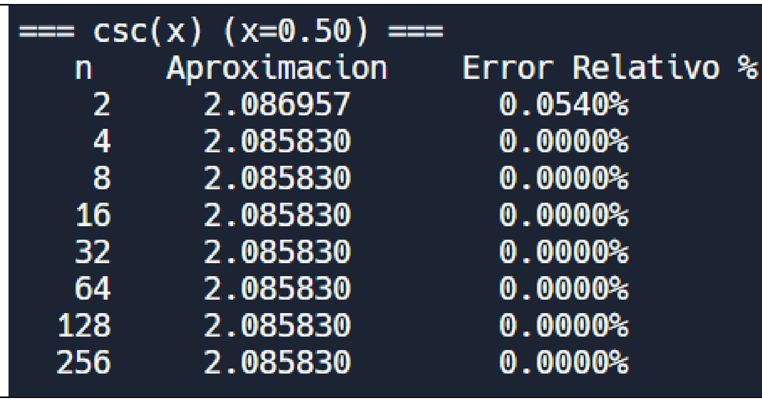
Tabla de Contenidos

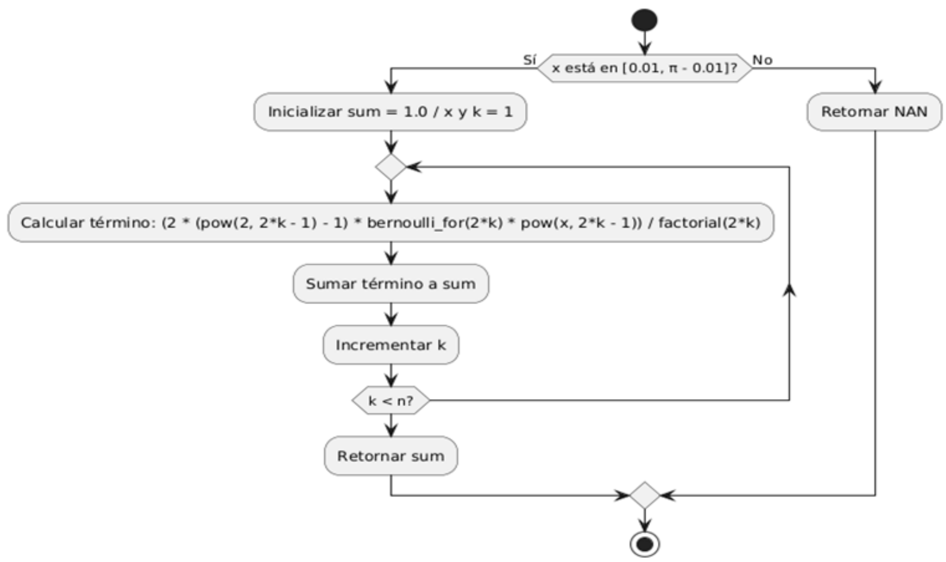
=== csc(x) (x=0.80) ===

n	Aproximacion	Error Relativo %
2	1.399254	0.3763%
4	1.394009	0.0001%
8	1.394008	0.0000%
16	1.394008	0.0000%
32	1.394008	0.0000%
64	1.394008	0.0000%
128	1.394008	0.0000%
256	1.394008	0.0000%

=== csc(x) (x=-0.30) ===

n	Aproximacion	Error Relativo %
2	-3.384095	0.0068%
4	-3.383863	0.0000%
8	-3.383863	0.0000%
16	-3.383863	0.0000%
32	-3.383863	0.0000%
64	-3.383863	0.0000%
128	-3.383863	0.0000%
256	-3.383863	0.0000%

<table>
<tr><td></td><td>

```
=== csc(x) (x=0.50) ===
  n     Aproximacion     Error Relativo %
  2       2.086957         0.0540%
  4       2.085830         0.0000%
  8       2.085830         0.0000%
 16       2.085830         0.0000%
 32       2.085830         0.0000%
 64       2.085830         0.0000%
128       2.085830         0.0000%
256       2.085830         0.0000%
```
</td></tr>
<tr><td>Diagrama</td><td>

</td></tr>
</table>

Función	$\operatorname{sen}^{-1}(x)$
Descripción	El algoritmo calcula una aproximación de la función arcsin(*x*) utilizando una serie de Taylor. En cada iteración del bucle for, se suman términos que dependen de potencias de *x* y coeficientes binomiales. El valor de *x* y el número de términos *n* son definidos por el usuario. La aproximación es válida solo para *x* en el intervalo *[-1,1]*
Código	

```
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include <stdlib.h>

#define MAX_ITER 1000
```

```c
bool validar_rango_x(double x, double min,
double max, const char* serie) {
    if (x < min || x > max) {
        printf("Error en %s: x debe estar en
[%.2f, %.2f]\n", serie, min, max);
        return false;
    }
    return true;
}

double arcsen_for(double x, int n) {
    if (!validar_rango_x(x, -1, 1, "arcsen(x)"))
return NAN;
    double sum = x, term = x;
    for (int k = 1; k < n; k++) {
        term *= x * x * (2*k - 1) / (2*k);
        sum += term / (2*k + 1);
    }
    return sum;
}

int main() {
    double x = 0.5;
    int n = 5;
    printf("arcsen(%.2f) aprox: %.6f (Real:
%.6f)\n", x, arcsen_for(x, n), asin(x));
    return 0;
}
```

Capturas

```
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output> & .\
arcsen(0.50) aprox: 0.523585 (Real: 0.523599)
PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output>
```

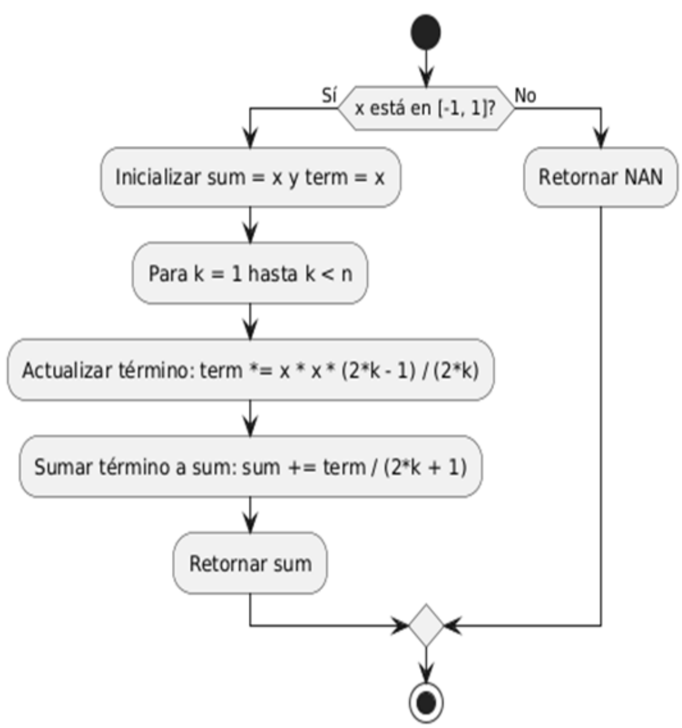
Tabla de Contenidos

Arsen(0.50)		
n	Aproximacion	Error Relativo
2	0.520833	0.5282
4	0.523526	0.0139
8	0.523599	0.0000
16	0.523599	0.0000
32	0.523599	0.0000
64	0.523599	0.0000
128	0.523599	0.0000
256	0.523599	0.0000

Arsen(0.80)		
n	Aproximacion	Error Relativo
2	0.885333	4.5252
4	0.919272	0.8653
8	0.926716	0.0624
16	0.927289	0.0007
32	0.927295	0.0000
64	0.927295	0.0000
128	0.927295	0.0000
256	0.927295	0.0000

Arsen(-0.30)		
n	Aproximacion	Error Relativo
2	-0.3045	0.0632
4	-0.304693	0.0000
8	-0.304693	0.0000
16	-0.304693	0.0000
32	-0.304693	0.0000
64	-0.304693	0.0000
128	-0.304693	0.0000
256	-0.304693	0.0000

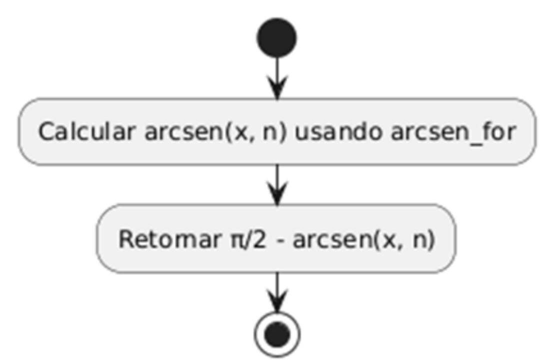
Diagrama



Función	$\cos^{-1}(x)$
Descripción	El algoritmo calcula una aproximación de la función arccos(x) utilizando la relación trigonométrica $arccos(x)=\pi 2-arcsin(x)$. Se basa en la función arcsen_for del ejercicio 24. El valor de x y el número de términos n son definidos por el usuario. La aproximación es válida solo para x en el intervalo $[-1,1]$

Código

```
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include <stdlib.h>

#define MAX_ITER 1000

bool validar_rango_x(double x, double min, double
max, const char* serie) {
    if (x < min || x > max) {
        printf("Error en %s: x debe estar en [%.2f,
%.2f]\n", serie, min, max);
        return false;
    }
    return true;
}

double arcsen_for(double x, int n) {
    if (!validar_rango_x(x, -1, 1, "arcsen(x)"))
return NAN;
    double sum = x, term = x;
    for (int k = 1; k < n; k++) {
        term *= x * x * (2*k - 1) / (2*k);
        sum += term / (2*k + 1);
    }
    return sum;
}

double arccos_for(double x, int n) {
    return M_PI/2 - arcsen_for(x, n);
}

int main() {
    double x = 0.7;
    int n = 5;
    printf("arccos(%.2f) aprox: %.6f (Real: %.6f)\n",
x, arccos_for(x, n), acos(x));
    return 0;
}
```

Capturas	PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output> & .\'serie2 arccos(0.70) aprox: 0.796122 (Real: 0.795399) PS C:\Users\Fernando Torres\OneDrive\Escritorio\Fundamentos\Practicas\output>
Tabla de Contenidos	=== arccos(x) (x=-0.30) === n Aproximacion Error Relativo % 2 1.875296 0.0103% 4 1.875488 0.0000% 8 1.875489 0.0000% 16 1.875489 0.0000% 32 1.875489 0.0000% 64 1.875489 0.0000% === arccos(x) (x=0.80) === n Aproximacion Error Relativo % 2 0.685463 6.5209% 4 0.651525 1.2469% 8 0.644080 0.0899% 16 0.643508 0.0010% 32 0.643501 0.0000% 64 0.643501 0.0000% === arccos(x) (x=0.50) === n Aproximacion Error Relativo % 2 1.049963 0.2641% 4 1.047270 0.0070% 8 1.047198 0.0000% 16 1.047198 0.0000% 32 1.047198 0.0000% 64 1.047198 0.0000%
Diagrama	

Función	$\tan^{-1}(x)$
Descripción	El algoritmo calcula una aproximación de la función arcotangente (arctan(x)) utilizando la serie de

	Taylor. En cada iteración del bucle do-while, se suman términos alternantes.

Código

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

#define MAX_ITER 1000

bool validar_rango_x(double x, double min, double max, const char* serie) {
    if (x < min || x > max) {
        printf("Error en %s: x debe estar en [%.2f, %.2f]\n", serie, min, max);
        return false;
    }
    return true;
}

double arctan_do_while(double x, int n) {
    if (!validar_rango_x(x, -1, 1, "arctan(x)")) return NAN;
    double sum = 0.0;
    int k = 0;
    double term = x; // Primer término: x
    sum = term;
    k++; // Empezar desde k=1
    do {
        term *= -x*x; // Alternar signo y actualizar término
        sum += term/(2*k + 1);
        k++;
    } while (k < n);
    return sum;
}

int main() {
    double x = 1.0; // x=1 es válido para arctan
    int n = 1000;
    printf("arctan(%.2f) aprox: %.6f (Real: %.6f)\n", x, arctan_do_while(x, n), atan(x));
    return 0;
}
```

Capturas

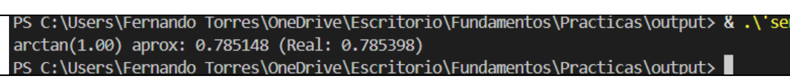
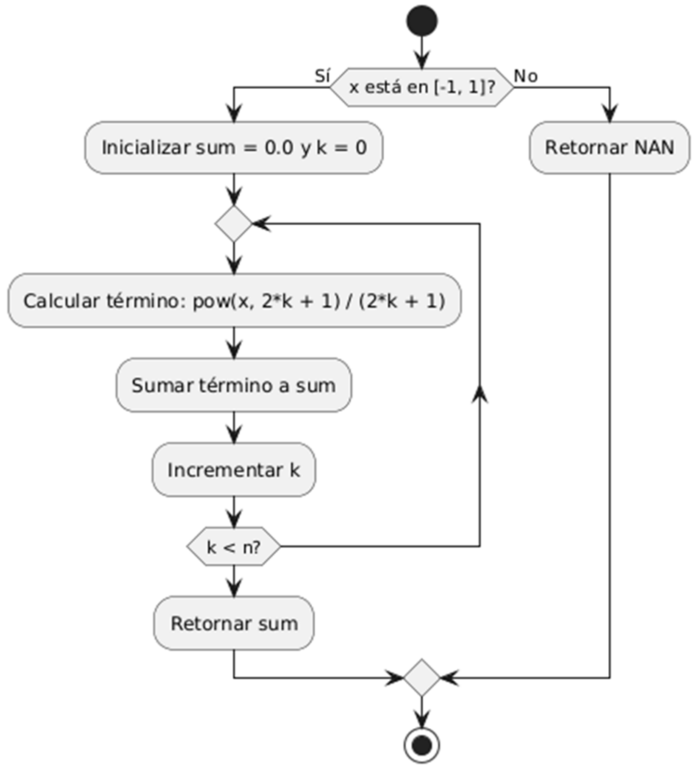


Tabla de Contenidos

Artan(0.50)		
n	Aproximacion	Error Relativo
2	0.541667	1.3908
4	0.549033	0.0498
8	0.549306	0.0001
16	0.549306	0.0000
32	0.549306	0.0000
64	0.549306	0.0000
128	0.549306	0.0000
256	0.549306	0.0000

Artan(-0.30)		
n	Aproximacion	Error Relativo
2	-0.309	0.1679
4	-0.30952	0.0000
8	-0.30952	0.0000
16	-0.30952	0.0000
32	-0.30952	0.0000
64	-0.30952	0.0000
128	-0.30952	0.0000
256	-0.30952	0.0000

Artan(0.80)		
n	Aproximacion	Error Relativo
2	0.970667	11.6461
4	1.066162	2.9538
8	1.095459	0.2871
16	1.098564	0.0044
32	1.098612	0.0000
64	1.098612	0.0000
128	1.098612	0.0000
256	1.098612	0.0000

Diagrama	

Funciones Hiperbólicas

Función	senh(x)
Descripción	El algoritmo aproxima mediante series de Taylor el valor del seno hiperbólico de X para un valor real de X ingresado, usando una cantidad N para un valor natural de términos ingresados, devolviendo el valor original de X y el resultado de la función hiperbólica con precisión de 12 decimales. Según las pruebas realizadas, el valor idóneo de N iteraciones que maximiza la precisión del algoritmo con respecto al valor teórico ideal se encuentra en el rango comprendido entre 64 y 128, habiendo casos donde el resultado se vuelve indeterminado si se utilizan 256 términos debido a la falta de precisión de las variables.
Código	/*Aproximar Seno hiperbólico de X Senh(x) Elaborado por Gustavo González De la Cruz*/

```
#include <stdio.h>
/*Se elaboraron funciones usadas frecuentemente con el fin
de
facilitar la comprensión y depuración del código*/

// Función para calcular X^n
double X_a_n(double X, int n) {
    double resu = 1.0;//Definir un resultado por defecto//
    for (int j = 0; j < n; j++) {
        resu *= X;//Multiplicar X por X N veces//
    }
    return resu;//Devolver el valor de X^N//
}

// Funcion para calcular el factorial de N
double factor(int N) {
    double resul = 1.0;//Definir un resultado por defecto//
    for (int k = 2; k <= N; k++) {
        resul *= k;//Multiplicar 1 por 2 por ... N-1 por
N//
    }
    return resul;//Devolver el valor de X!//
}

//Algoritmo principal//
int main() {
    //Declarar variables//
    double X=0, res=0;
    int n=0, i;
    printf("\nSeno hiperbolico de X\n");//Mostrar al
usuario la función hiperbólica que se calculará//
    printf("\nIngrese el valor de X: ");//Solicitar el
valor de X al usuario//
    scanf("%lf", &X);//Leer el valor de X ingresado//
    res = X;//Asignar al resultado de X para ahorrar una
iteración//
    do {
//Solicitar al usuario que la cantidad de
iteraciones/precisión deseada//
        printf("\nIngrese el numero de iteraciones con las
que desea aproximar Sinh(x): ");
        scanf("%d", &n);//Leer el numero de iteraciones
ingresados//
    } while (n < 1);//Mientras el valor sea menor a uno//

//Calcular Senh(x)//
    for (i = 1; i <= n; i++) {
        res += (1.0 / (factor((2*i)+1)))*(X_a_n(X,
(2*i)+1));/*Sumar X elevado al doble del numero de
iteración mas uno
                                    dividido entre el
factorial del doble del numero de iteración mas uno*/
    }
```

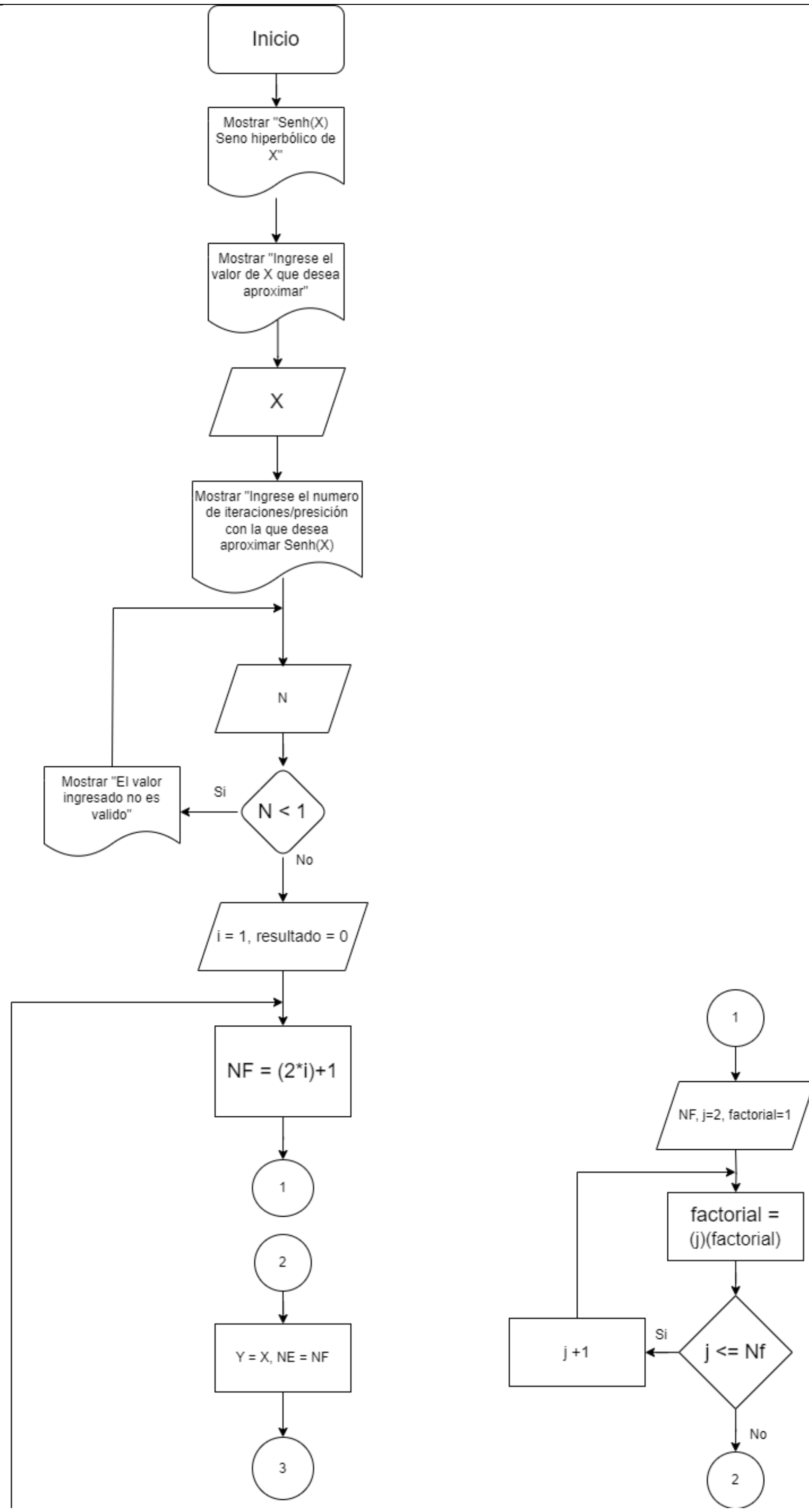
```
    //Imprimir al usuario el valor de X ingresado y el
resultado de la aproximación//
    printf("\nEl resultado del Seno hiperbolico de %0.5lf
es: %0.12lf", X, res);
    return 0;
}
```

Capturas

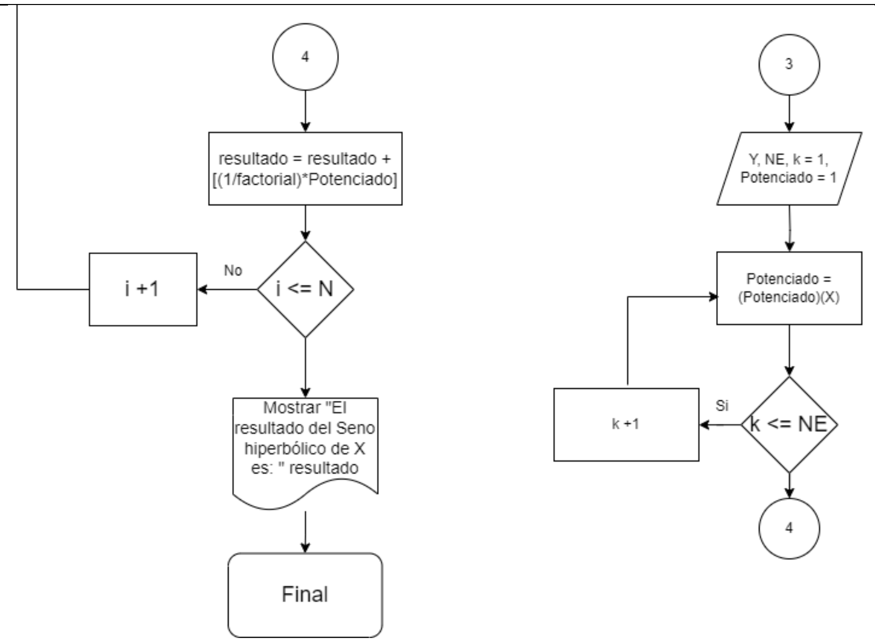
```
Microsoft Windows [Versión 10.0.19045.5555]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Gustavo\Documents\C++\Prog Ing P1>cd "c:\Users\Gustavo\Documents\C++\Prog Ing P1\output"

c:\Users\Gustavo\Documents\C++\Prog Ing P1\output>.\"Senh_X.exe"

Seno hiperbolico de X

Ingrese el valor de X: 2.216

Ingrese el numero de iteraciones con las que desea aproximar Sinh(x): 64

El resultado del Seno hiperbolico de 2.21600 es: 4.530765343830
c:\Users\Gustavo\Documents\C++\Prog Ing P1\output>
```

Tabla de Contenidos

$Senh(X)$	Senh(2.216)		Senh(3.504)		Senh(9.704)	
n	Valor	Error	Valor	Error	Valor	Error
2	4.4749841	0.01231166	15.0762519	0.09228698	879.0928177	0.89268237
4	4.5306017	3.61184E-05	16.5825119	0.00159788	4589.6477998	0.43970634
8	4.5307665	2.55182E-07	16.6090509	1.23E-08	8132.3533380	0.00722099
16	4.5307653	9.6738E-09	16.6090511	2.5835E-10	8191.5041367	4.5504E-10
32	4.5307653	9.6738E-09	16.6090511	2.5835E-10	8191.5041404	3.3515E-12
64	4.5307653	9.6738E-09	16.6090511	2.5835E-10	8191.5041404	3.3515E-12
128	4.5307653	9.6738E-09	16.6090511	2.5835E-10	8191.5041404	3.3515E-12
256	4.5307653	9.6738E-09	16.6090511	2.5835E-10	-1.#IND0000	1

Diagrama	

Inicio
Mostrar "Senh(X) Seno hiperbólico de X"
Mostrar "Ingrese el valor de X que desea aproximar"
X
Mostrar "Ingrese el numero de iteraciones/presición con la que desea aproximar Senh(X)
N
Mostrar "El valor ingresado no es valido"
Si
N < 1
No
i = 1, resultado = 0
NF = (2*i)+1
1
2
Y = X, NE = NF
3
1
NF, j=2, factorial=1
factorial = (j)(factorial)
j +1
Si
j <= Nf
No
2


4
resultado = resultado + [(1/factorial)*Potenciado]
No
i +1
i <= N
Mostrar "El resultado del Seno hiperbólico de X es: " resultado
Final
3
Y, NE, k = 1, Potenciado = 1
Potenciado = (Potenciado)(X)
Si
k +1
k <= NE
4

Función	cosh(x)
Descripción	El algoritmo aproxima mediante series de Taylor el valor del coseno hiperbólico de X para un valor real de X ingresado, usando una cantidad N para un valor natural de términos ingresados, devolviendo el valor original de X y el resultado de la función hiperbólica con precisión de 12 decimales. Según las pruebas realizadas, el valor idóneo de N iteraciones que maximiza la precisión del algoritmo con respecto al valor teórico ideal se encuentra en el rango comprendido entre 32 y 128, ya que el resultado se indetermina en ciertos casos al usar valores cercanos a 256 debido a la imprecisión de las variables.
Código	(ver código abajo)

```
/*Aproximar Coseno hiperbólico de X
Cosh(x)
Elaborado por Gustavo González De la Cruz*/
#include <stdio.h>
/*Se elaboraron funciones usadas frecuentemente con el
fin de
facilitar la comprensión y depuración del código*/

// Función para calcular X^n//
double X_a_n(double X, int n) {
    double resu = 1.0;//Definir un resultado por
defecto//
    for (int j = 0; j < n; j++) {
        resu *= X;//Multiplicar X por X N veces//
    }
    return resu;//Devolver el valor de X^N//
}

// Función para calcular el factorial de N//
double factor(int N) {
    double resul = 1.0;//Definir un resultado por
defecto//
    for (int k = 2; k <= N; k++) {
        resul *= k;//Multiplicar 1 por 2 por ... hasta
N//
    }
    return resul;//Devolver el valor de X!//
}

//Algoritmo principal//
int main()
{
```

```
    //Declaración de variables//
    double res =1, termino, X=0;
    long int n=0, i;

    printf("\nCosh(x) Coseno hiperbolico de
X\n");//Mostrar al usuario la función hiperbolica que se
calculará//
    printf("\nIngrese el valor de X: ");//Solicitar al
usuario un valor de X//
    scanf("%lf", &X);
    do {//Solicitar al usuario el numero de
iteraciones/precisión con la desea aproximar el valor de
Cosh(x)//
        printf("\nIngrese el numero de iteraciones con
las que desea aproximar Cosh(x): ");
        scanf("%ld", &n);
    } while (n < 1);//Hasta que ingrese un valor
diferente a 0//
    //Calcular Cosh(x) usando la serie de Taylor
    for ( i = 1; i <= n; i++)
    {
        /*Definir el termino actual de la serie como uno
entre el factorial del numero de iteración actual
        multiplicado por X elevado al doble del numero de
iteración actual*/
        termino = (1/factor(2*i))*(X_a_n(X, 2*i));
        res += termino;//Sumar el valor obtenido al
resultado que se mostrará//
    }
    printf("\nEl Coseno hiperbolico de %0.5lf es :
%0.12lf", X, res);//Mostrar al usuario el valor de X y
del Cosh(x) obtenido//
    return 0;
}
```

Capturas

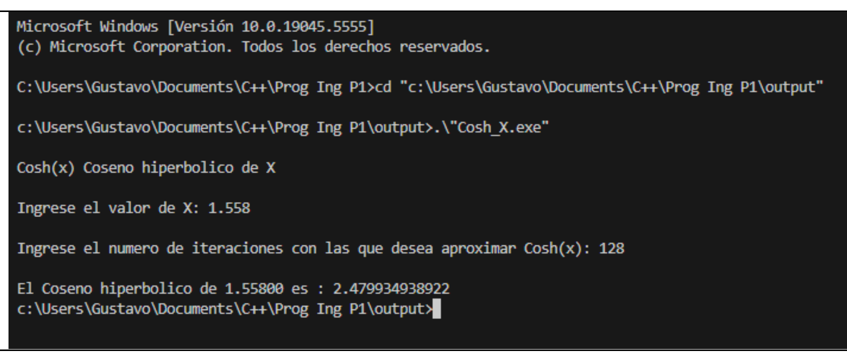
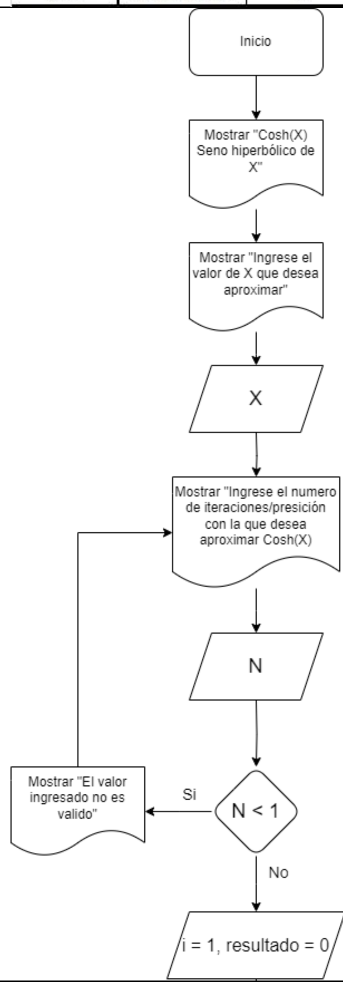
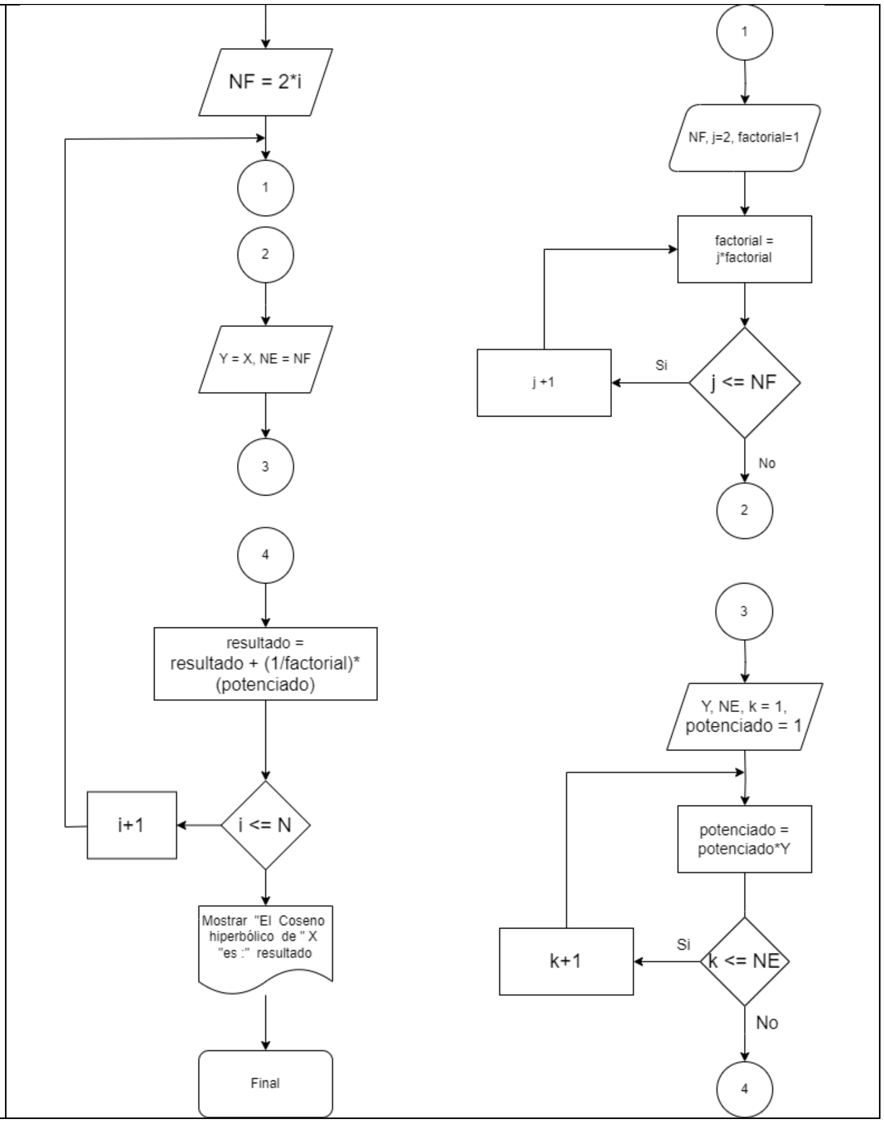


Tabla de Comparación

$Cosh(X)$	Cosh(9.698)		Cosh(1.558)		Cosh(4.371)	
n	Valor	Error	Valor	Error	Valor	Error
2	416.5934759	0.948837168	2.4591859	0.008366767	25.7622171	0.34890751
4	3512.6725193	0.568600367	2.4799112	9.5724E-06	38.7530462	0.02058828
8	8024.8748296	0.014446112	2.4799349	1.56947E-08	39.5676209	1.411E-06
16	8142.5023163	1.59006E-09	2.4799349	1.56947E-08	39.5676767	7.8231E-10
32	8142.5023292	5.77553E-12	2.4799349	1.56947E-08	39.5676767	7.8231E-10
64	8142.5023292	5.77553E-12	2.4799349	1.56947E-08	39.5676767	7.8231E-10
128	8142.5023292	5.77553E-12	2.4799349	1.56947E-08	39.5676767	7.8231E-10
256	-1.#IND0000	1	2.4799349	1.56947E-08	-1.#IND0000	1

Diagrama



Función	$\tanh(x)$
Descripción	El algoritmo aproxima mediante series de Taylor el valor de la Tangente hiperbólica de X para un valor real de X ingresado cuyo valor absoluto sea menor a Pi medios, usando una cantidad N para un valor natural de términos ingresados, devolviendo el valor original de X y el resultado de la función hiperbólica con precisión de 12 decimales, esto lo realiza aprovechando la identidad de la función Tangente hiperbólica de X: Seno hiperbólico de X entre Coseno hiperbólico de X. Según las pruebas realizadas, el valor idóneo de N iteraciones que maximiza la precisión del algoritmo con respecto al valor teórico ideal se encuentra en el rango comprendido entre 8 y 256,

	por lo que aún con pocos términos es posible obtener resultados aceptables.
Código	(see code below)

```
/*Aproximar Seno hiperbólico de X
Tanh(x)
Elaborado por Gustavo González De la Cruz*/
#include <stdio.h>
/*Se elaboraron funciones usadas frecuentemente con el fin de
facilitar la comprensión y depuración del código*/

// Función para calcular X^n
double X_a_n(double X, int n) {
    double resu = 1.0;//Definir un resultado por defecto//
    for (int j = 0; j < n; j++) {
        resu *= X;//Multiplicar X por X N veces//
    }
    return resu;//Devolver el valor de X^N//
}

// Funcion para calcular el factorial de N//
double factor(int N) {
    double resul = 1.0;//Definir un resultado por defecto//
    for (int k = 2; k <= N; k++) {
        resul *= k;//Multiplicar 1 por 2 por ... N-1 por N//
    }
    return resul;//Devolver el valor de X!//
}

//Algoritmo principal//
int main()
{
    //Declarar variables//
    double X, res1, res2=1;
    long int n, i;

    //Mostrar al usuario la función hiperbólica que se
realizará//
    printf("\nTangente hiperbolica de X\n");
    do
    {//Solicitar el valor de X al usuario//
        printf("\nIngrese un valor de |X| < pi/2: ");
        scanf("%lf", &X);//Leer el valor de X ingresado//
        if (X>=1.570796326794896||X<=-
1.570796326794896){//Comprar el valor de X con pi/2 //
            printf("\nEl valor ingresado no es valido, intente de
nuevo");//Indicar al usuario que el valor no es valido//
            X = 2;//Igualar X a dos para repetir el ciclo Do
While//
        }
    } while (X>=1.570796326794896||X<=-
1.570796326794896);//Mientras el valor absoluto de X sea mayor a
pi medios//
```

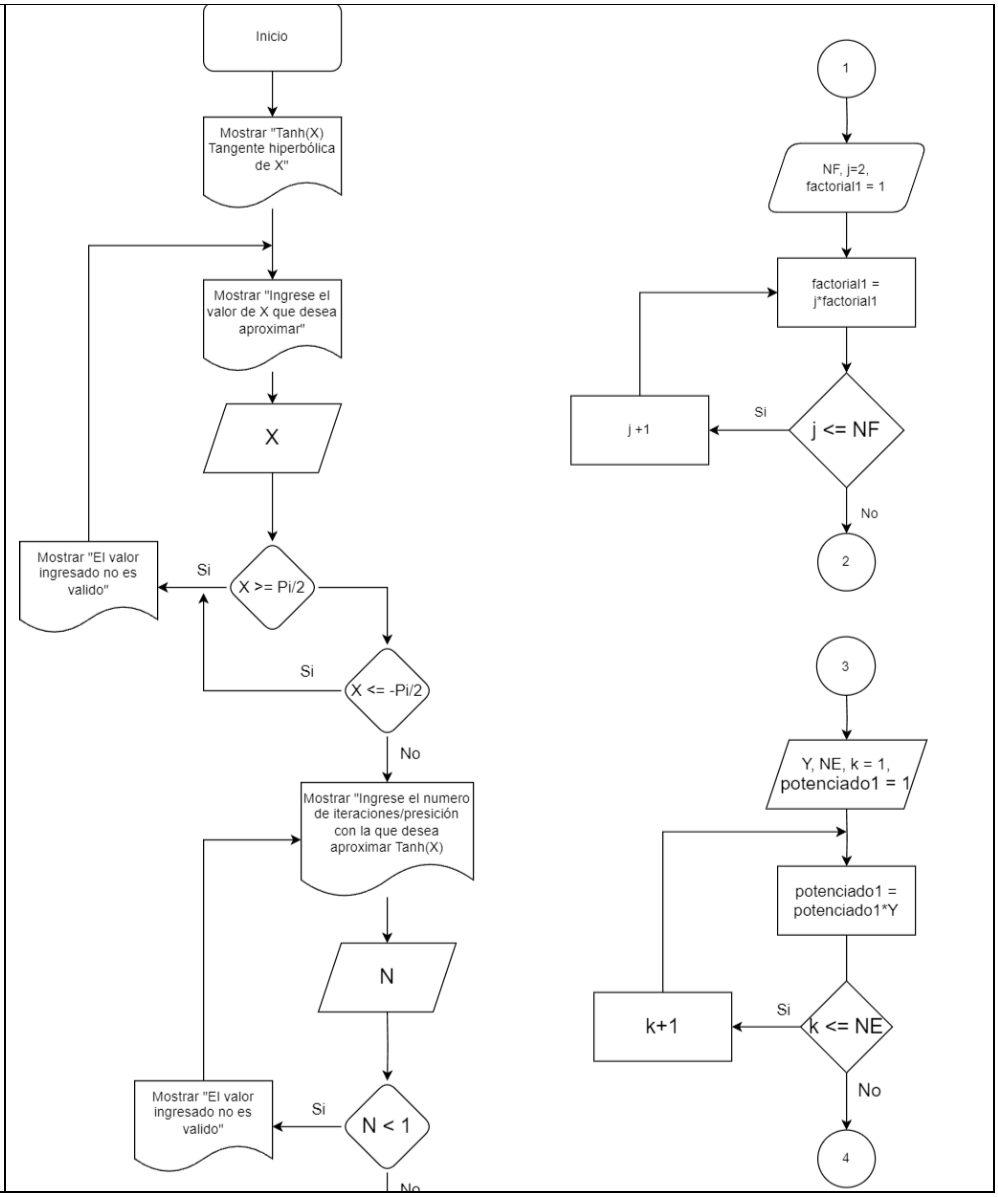
```
    res1 = X;//Igualar el resultado del Sen(X) a X, para ahorrar
iteraciones//
    do
    {//Solicitar al usuario que ingrese el numero de
iteraciones/precisión que desea//
        printf("\nIngrese el numero de iteraciones con las que
desea aproximar Tanh(x): ");
        scanf("%ld", &n);//Leer el valor ingresado por el
usuario//
    } while (n < 1);//Mientras el valor ingresado sea menor a
uno//

    for (i = 1; i <= n; i++)
    {
    //Calcular Senh(x)//
                /*Sumar al resultado de Seno la división de uno
entre el factorial del doble del numero de iteración mas uno
                y multiplicarlo por X elevado a la potencia del
doble del numero de iteración mas uno*/
        res1 += (1.0 /(factor((2*i)+1)))*(X_a_n(X, (2*i)+1));
    //Calcular Cosh(x)//
                /*Sumar al resultado de Coseno la división uno
entre el factorial del doble del numero de iteración
                multiplicado por X elevado a la potencia del
doble del numero de iteración*/
        res2 += (1.0/factor(2*i))*(X_a_n(X, 2*i));
    }
    //Mostrar al usuario el valor de X ingresado y el resultado
de dividir Senh X entre Cosh X//
    printf("\nLa Tanh(%0.5lf) es %0.12lf", X, (res1/res2));
    return 0;
}
```

Capturas

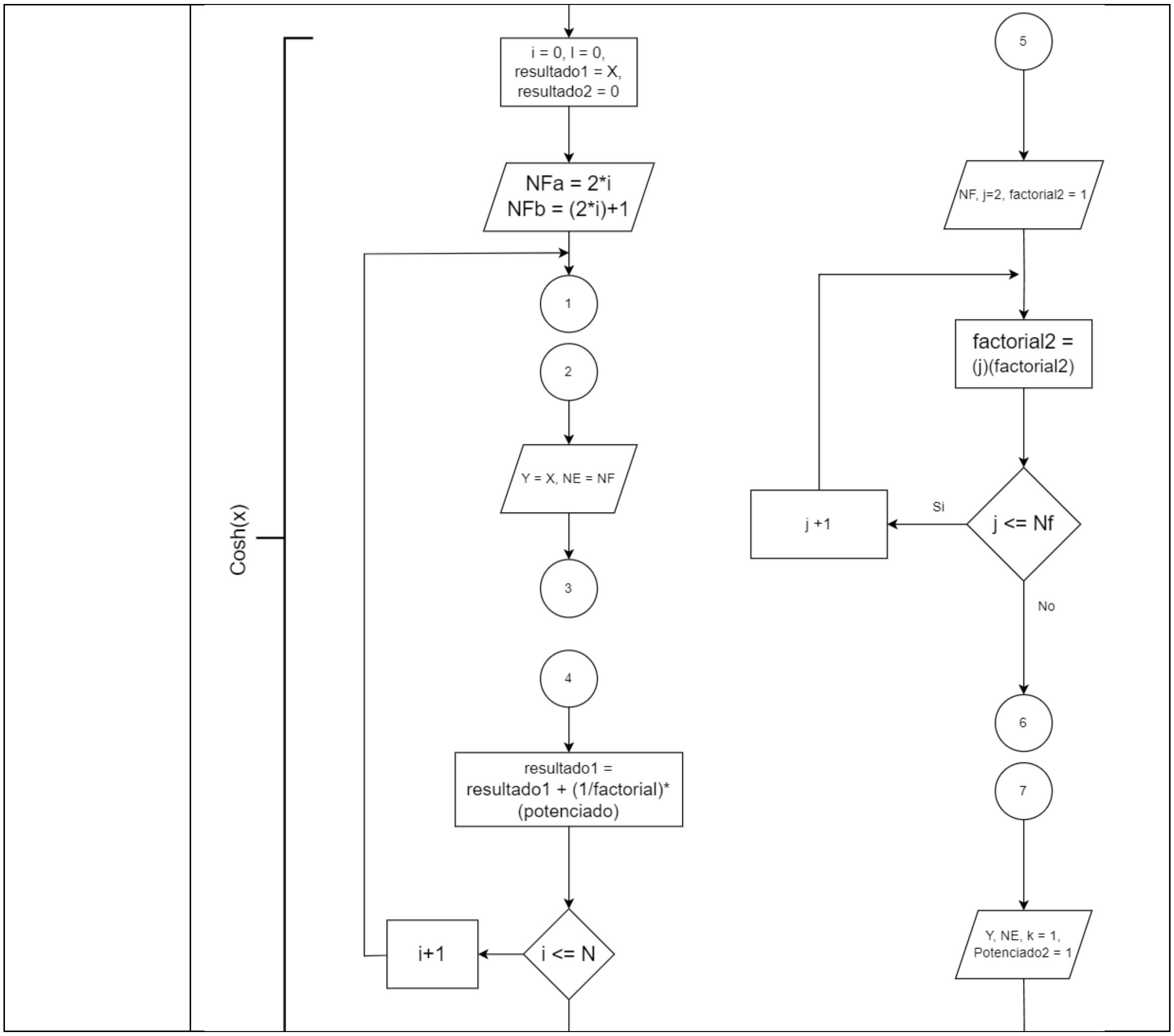
```
Microsoft Windows [Versión 10.0.19045.5555]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Gustavo\Documents\C++\Prog Ing P1>cd "c:\Users\Gustavo\Documents\C++\Prog Ing P1\output"

c:\Users\Gustavo\Documents\C++\Prog Ing P1\output>.\"Tanh_X.exe"

Tangente hiperbolica de X

Ingrese un valor de |X| < pi/2: 0.623

Ingrese el numero de iteraciones con las que desea aproximar Tanh(x): 8

La Tanh(0.62300) es 0.553213348136
c:\Users\Gustavo\Documents\C++\Prog Ing P1\output>
```

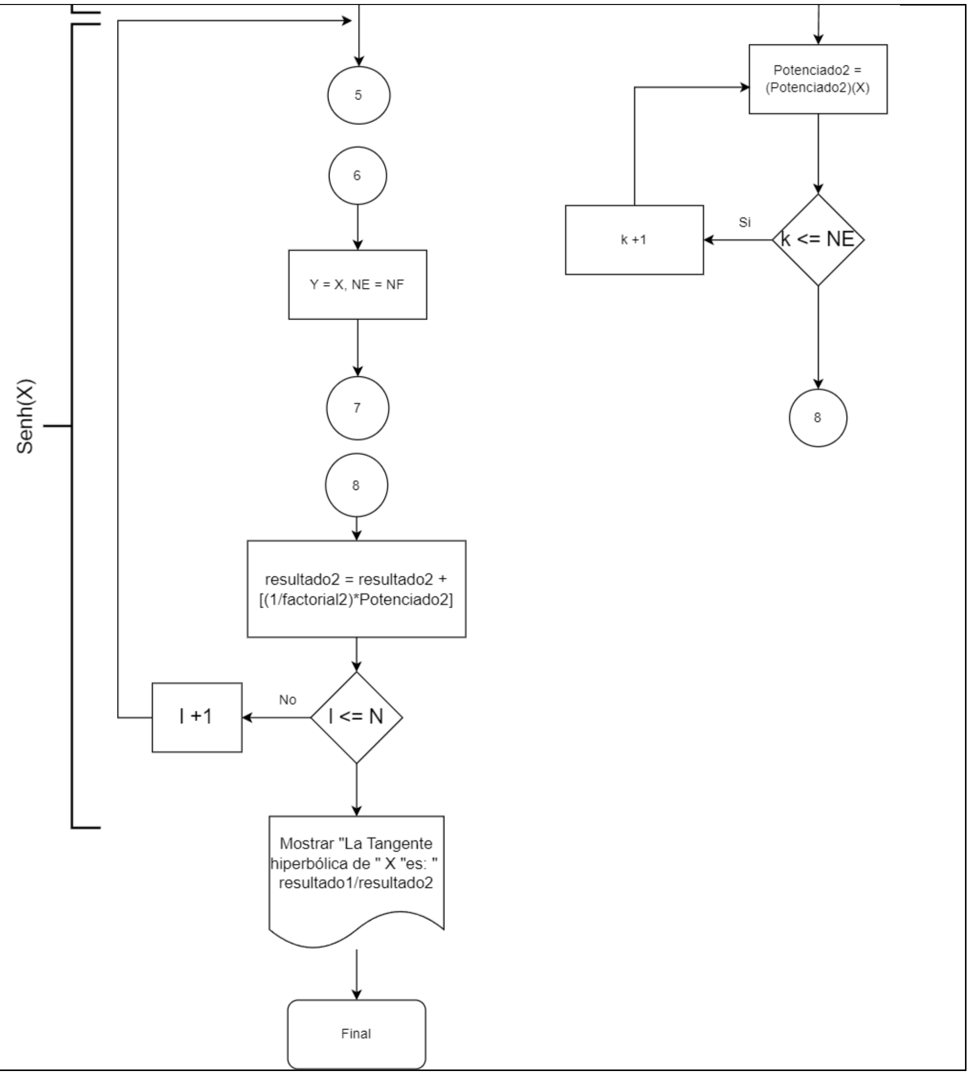
Tabla de Contenidos

$Tanh(X)$	Tanh(0.148)		Tanh(-1.346)		Tanh(0.623)	
n	Valor	Error	Valor	Error	Valor	Error
2	0.1469287	5.93665E-07	-0.8759553	0.003263571	0.5532449	5.7034E-05
4	0.1469287	5.93665E-07	-0.8731078	2.22525E-06	0.5532133	8.7012E-08
8	0.1469287	5.93665E-07	-0.8731058	6.54208E-08	0.5532133	8.7012E-08
16	0.1469287	5.93665E-07	-0.8731058	6.54208E-08	0.5532133	8.7012E-08
32	0.1469287	5.93665E-07	-0.8731058	6.54208E-08	0.5532133	8.7012E-08
64	0.1469287	5.93665E-07	-0.8731058	6.54208E-08	0.5532133	8.7012E-08
128	0.1469287	5.93665E-07	-0.8731058	6.54208E-08	0.5532133	8.7012E-08
256	0.1469287	5.93665E-07	-0.8731058	6.54208E-08	0.5532133	8.7012E-08

Diagrama	

Inicio
Mostrar "Tanh(X) Tangente hiperbólica de X"
Mostrar "Ingrese el valor de X que desea aproximar"
X
X >= Pi/2
Si
Mostrar "El valor ingresado no es valido"
X <= -Pi/2
Si
No
Mostrar "Ingrese el numero de iteraciones/presición con la que desea aproximar Tanh(X)
N
N < 1
Si
Mostrar "El valor ingresado no es valido"
No
1
NF, j=2, factorial1 = 1
factorial1 = j*factorial1
j <= NF
Si
j +1
No
2
3
Y, NE, k = 1, potenciado1 = 1
potenciado1 = potenciado1*Y
k <= NE
Si
k+1
No
4


Cosh(x)
i = 0, l = 0,
resultado1 = X,
resultado2 = 0
NFa = 2*i
NFb = (2*i)+1
1
2
Y = X, NE = NF
3
4
resultado1 =
resultado1 + (1/factorial)*
(potenciado)
i <= N
i+1
5
NF, j=2, factorial2 = 1
factorial2 =
(j)(factorial2)
j <= Nf
Si
j +1
No
6
7
Y, NE, k = 1,
Potenciado2 = 1

Función	$\operatorname{senh}^{-1}(x)$
Descripción	El algoritmo aproxima mediante series de Taylor el valor del seno hiperbólico inverso de X para un valor real de X ingresado cuyo valor absoluto sea menor a uno, usando una cantidad N para un valor natural de términos ingresados, devolviendo el valor original de X y el resultado de la función hiperbólica con precisión de 12 decimales. Según las pruebas realizadas, el valor idóneo de N iteraciones que maximiza la precisión del algoritmo con respecto al valor teórico ideal se encuentra alrededor de 64, siendo que valores menores mas cercanos a 32 ofrecen menor precisión, y valores

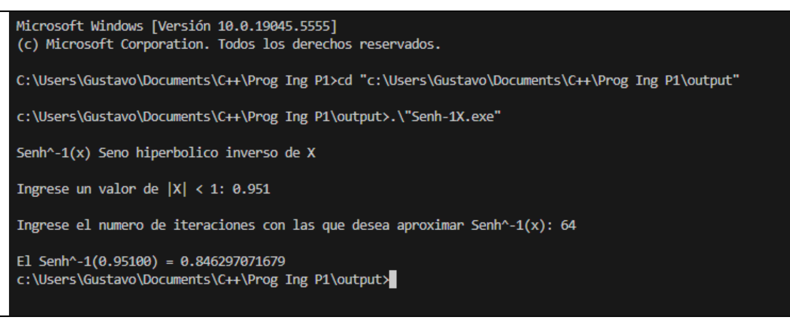
	mayores cercanos a 128 provocan la indeterminación del reultado.

Código

```c
/*Aproximar Seno hiperbólico inverso de X
Senh-1 (x)
Elaborado por Gustavo González De la Cruz*/
#include <stdio.h>
/*Se elaboraron funciones usadas frecuentemente con
el fin de
facilitar la comprensión y depuración del código*/

// Función para calcular X^n
double X_a_n(double X, long int n) {
    double resu = 1.0;//Definir un resultado por
defecto//
    for (int j = 0; j < n; j++) {
        resu *= X;//Multiplicar X por X N veces//
    }
    return resu;//Devolver el valor de X^N//
}

// Funcion para calcular el factorial de N//
double factor(long int N) {
    double resul = 1.0;//Definir un resultado por
defecto//
    for (int k = 2; k <= N; k++) {
        resul *= k;//Multiplicar 1 por 2 por ... N-1
por N//
    }
    return resul;//Devolver el valor de X!//
}
//Algoritmo principal//
int main()
{
    //Declarar variables//
    double X=0, res;
    long int i, n=0;
    printf("\nSenh^-1(x) Seno hiperbolico inverso de
X\n");//Mostrar al usuario la función hiperbólica que
se calculará//
    do
    {//Solicitar al usuario que ingrese un numero
cuyo valor absoluto sea menor a uno//
        printf("\nIngrese un valor de |X| < 1: ");
        scanf("%lf", &X);//Leer el valor ingresado//
        if (X >= 1.0||X <= -1.0)//Comparar el valor
de X con uno y menos uno//
        {//Si el valor de X es mayor a uno o menor a
menos uno//
            printf("\nEl valor ingresado no es
valido, intente de nuevo");//Mostrar que el valor
ingresado no es valido//
```

```
            X = 1;//Igualar X a uno//
        }
    } while (X==1);//Mientras X sea igual a uno//
    res = X;//Asignar el valor de X al resultado,
para ahorrar una iteración//
    do
    {//Solicitar al usuario el numero de
iteraciones/presición con la que quiere aproximar
Senh-1 (x)//
        printf("\nIngrese el numero de iteraciones
con las que desea aproximar Senh^-1(x): ");
        scanf("%ld", &n);//Leer el valor ingresado//
    } while (n < 1);//Mientras sea menor a uno//
    //Calcular el Senh-1 (x)//
    for ( i = 1; i < n; i++)
    {
        /*Sumar al resultado la multiplicación del
signo, definido según el numero de iteración
        multiplicado por el factorial del doble del
numero de iteracion,
        multiplicado por el valor de X elevado al
doble del numero de iteración mas uno,
        todo esto dividido entre la multiplicación de
cuatro elevado a la potencia del numero de iteración
        por el cuadrado del factorial del numero de
iteración por el doble del numero de iteración mas
uno*/
        res += ((i%2)?-1:1)*(((factor(2*i))*(X_a_n(X,
(2*i)+1))))/(((X_a_n(4,
i))*(factor(i))*(factor(i))*((2*i)+1)));
    }
    //Mostrar al usuario el valor de X ingresado y el
resultado de la operación//
    printf("\nEl Senh^-1(%0.5lf) = %0.12lf", X, res);
    return 0;
}
```

Capturas

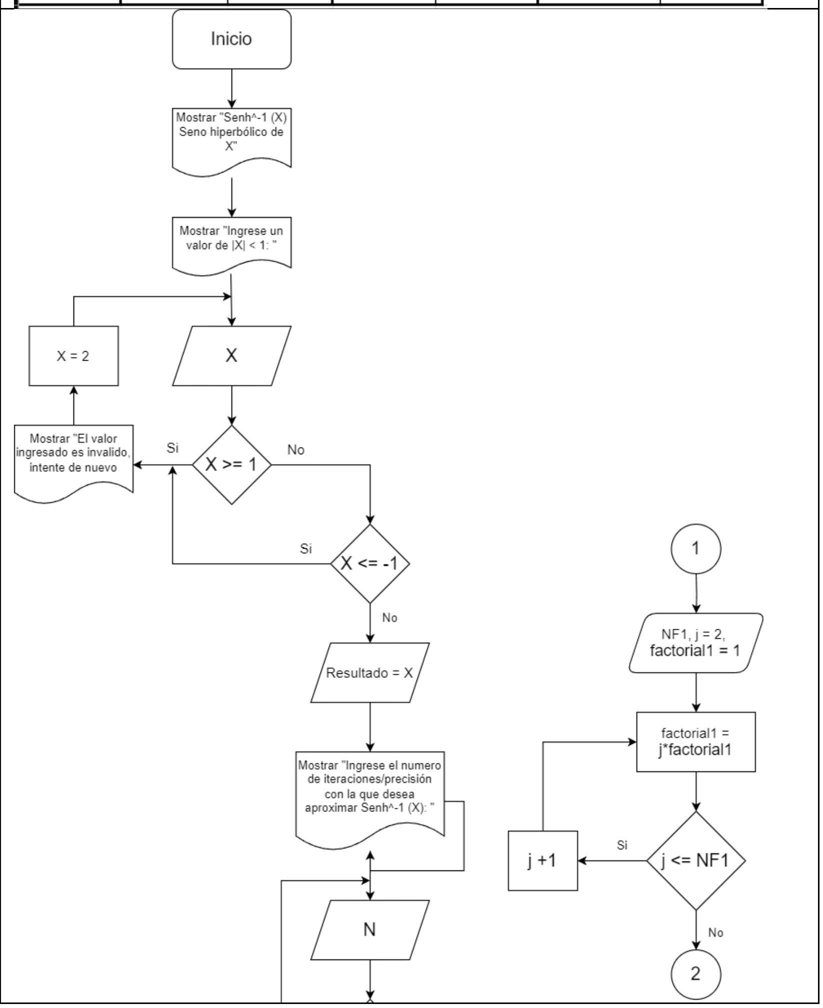


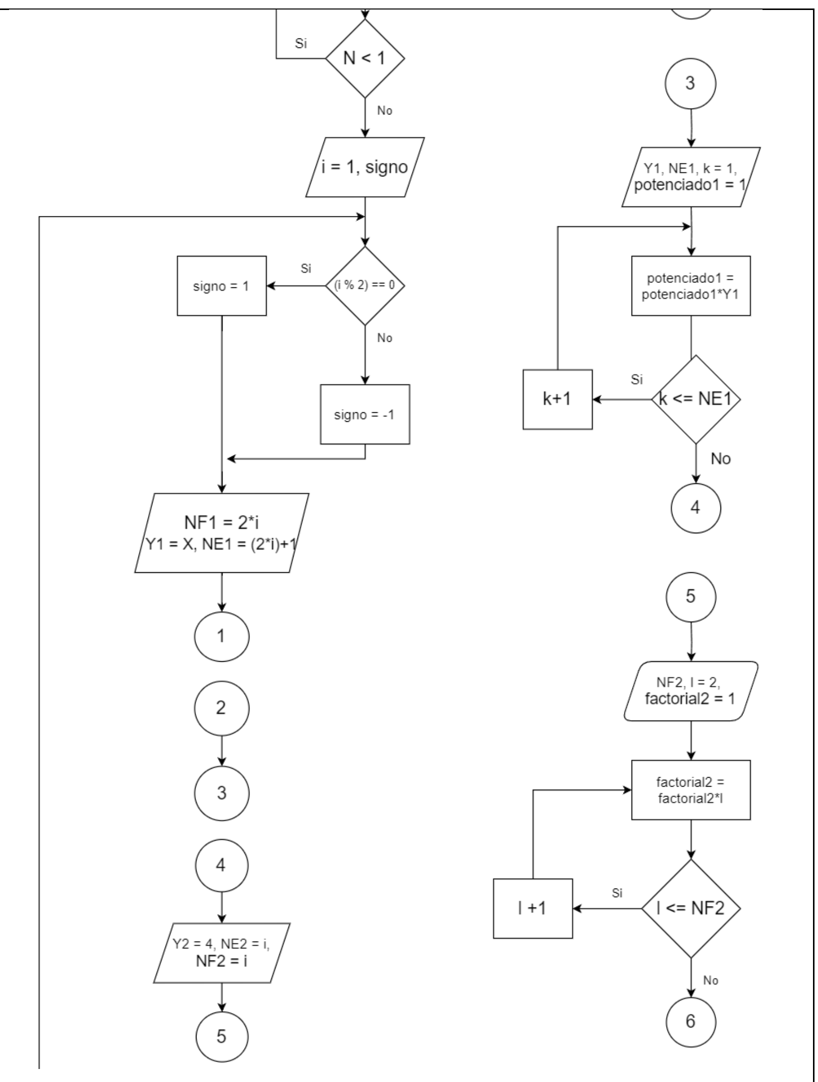
```
Microsoft Windows [Versión 10.0.19045.5555]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Gustavo\Documents\C++\Prog Ing P1>cd "c:\Users\Gustavo\Documents\C++\Prog Ing P1\output"

c:\Users\Gustavo\Documents\C++\Prog Ing P1\output>.\"Senh-1X.exe"

Senh^-1(x) Seno hiperbolico inverso de X

Ingrese un valor de |X| < 1: 0.951

Ingrese el numero de iteraciones con las que desea aproximar Senh^-1(x): 64

El Senh^-1(0.95100) = 0.846297071679
c:\Users\Gustavo\Documents\C++\Prog Ing P1\output>
```

Tabla de Contenidos

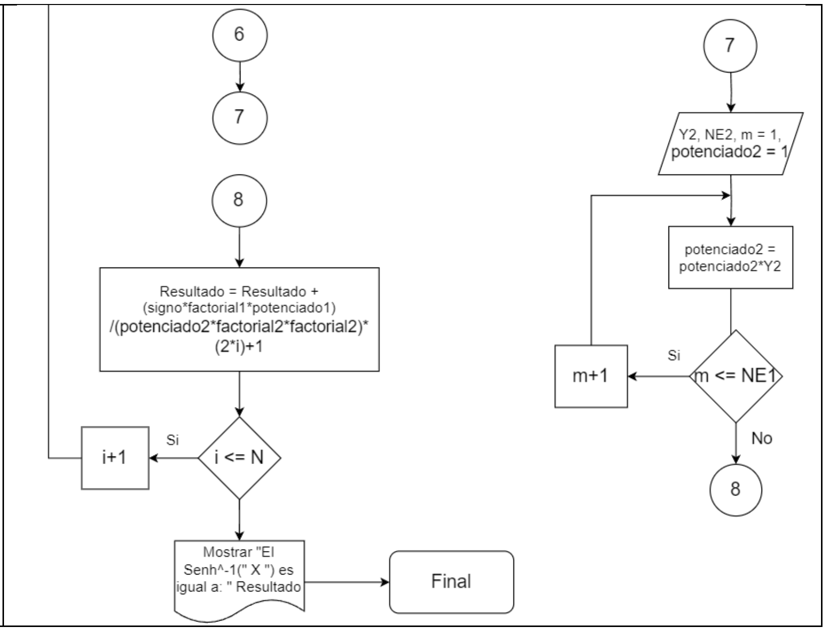
$Senh^{-1}(X)$	Senh^-1(0.106)		Senh^-1(-0.817)		Senh^-1(0.951)	
n	Valor	Error	Valor	Error	Valor	Error
2	0.1058014	1.03432E-05	-0.7261102	0.02651585	0.8076524	0.04566375
4	0.1058024	8.91669E-07	-0.7425638	0.00445677	0.8345858	0.01383877
8	0.1058024	8.91669E-07	-0.7456497	0.00031956	0.8435007	0.00330477
16	0.1058024	8.91669E-07	-0.7458847	4.4955E-06	0.8458549	0.000523
32	0.1058024	8.91669E-07	-0.74588805	4.2243E-09	0.8462662	3.7003E-05
64	0.1058024	8.91669E-07	-0.74588805	4.2243E-09	0.84629707	5.264E-07
128	-1.#IND0001	1	-1.#IND0000	1	-1.#IND0000	1
256	-1.#IND0000	1	-1.#IND0000	1	-1.#IND0001	1

Diagrama





Si
N < 1
No
i = 1, signo
Si
(i % 2) == 0
signo = 1
No
signo = -1
NF1 = 2*i
Y1 = X, NE1 = (2*i)+1
1
2
3
4
Y2 = 4, NE2 = i,
NF2 = i
5
3
Y1, NE1, k = 1,
potenciado1 = 1
potenciado1 =
potenciado1*Y1
Si
k+1
k <= NE1
No
4
5
NF2, l = 2,
factorial2 = 1
factorial2 =
factorial2*l
Si
l +1
l <= NF2
No
6

Función	$\tanh^{-1}(x)$
Descripción	El algoritmo aproxima mediante series de Taylor el valor de la Tangente hiperbólica inversa de X para un valor real de X ingresado cuyo valor absoluto sea menor a uno, usando una cantidad N para un valor natural de términos ingresados, devolviendo el valor original de X y el resultado de la función hiperbólica con precisión de 12 decimales. Según las pruebas realizadas, el valor idóneo de N iteraciones que maximiza la precisión del algoritmo con respecto al valor teórico ideal se encuentra en el rango comprendido entre 8 y 256, por lo que aún con pocos términos es posible obtener resultados aceptables. Sin embargo, existen casos particulares, como aquellos donde el valor de X se acerca a 1, donde se requiere un numero de iteraciones cercano a las 256 para obtener un valor con bajo error relativo.
Código	

```
/*Aproximar Seno hiperbólico de X
Tanh-1 (x)
Elaborado por Gustavo González De la Cruz*/
#include <stdio.h>
/*Se elaboraron funciones usadas frecuentemente con el fin de
facilitar la comprensión y depuración del código*/

// Función para calcular X^n
double X_a_n(double X, int n) {
```

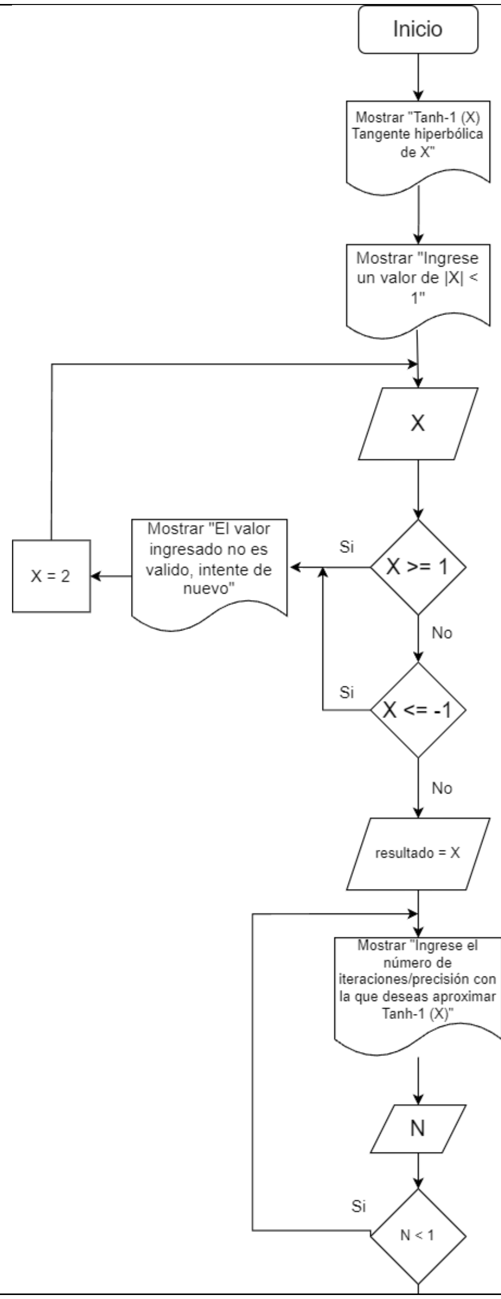
```c
    double resu = 1.0;//Definir un resultado por defecto//
    for (int j = 0; j < n; j++) {
        resu *= X;//Multiplicar X por X N veces//
    }
    return resu;//Devolver el valor de X^N//
}

//Algoritmo principal//
int main()
{
    //Declarar variables//
    double X=0, res;
    long int i, n=0;
    printf("\nTanh^-1(x) Tangente hiperbolica inversa de 
X\n");//Mostrar al usuario la función hiperbólica que se 
calculará//
    do{//Solicitar el valor de X al usuario//
        printf("\nIngrese un valor de |X| < 1: ");
        scanf("%lf", &X);//Leer el valor ingresado de X//
        if (X >= 1.0||X <= -1.0)//Comparar el valor de X con uno 
y menos uno//
        {//Si el valor ingresado es mayor a uno o menor a menos 
uno//
            printf("\nEl valor ingresado no es valido, intente de 
nuevo");//Indicar al usuario que el valor ingresado es invalido//
            X = 1;//Igualar X a dos//
        }
    } while (X==1);//Mientras X sea igual a dos//
    res = X;//Igualar el resultado a X, para ahorrar 
iteraciones//
    do{//Solicitar al usuario que ingrese el numero de 
iteraciones/presición deseada//
        printf("\nIngrese el numero de iteraciones con las que 
desea aproximar Tanh^-1(x): ");
        scanf("%ld", &n);//Leer el valor ingresado//
    } while (n < 1);//Mientras el numero de iteraciones sea menor 
a 1//
    //Calcular la tangente hiperbólica inversa de X//
    for ( i = 1; i <= n; i++)
    {
        //Sumar al resultado la división de X elevado al doble 
del numero de iteración mas uno//
        res += (X_a_n(X, (2*i)+1))/(float)(2*i+1);//Dividido 
entre el doble del numero de iteración mas uno//
    }
    //Mostrar el valor de X ingresado y el resultado aproximado//
    printf("\nLa Tanh^-1(%0.5lf) = %0.12lf", X, res);
    return 0;
}
```

Capturas

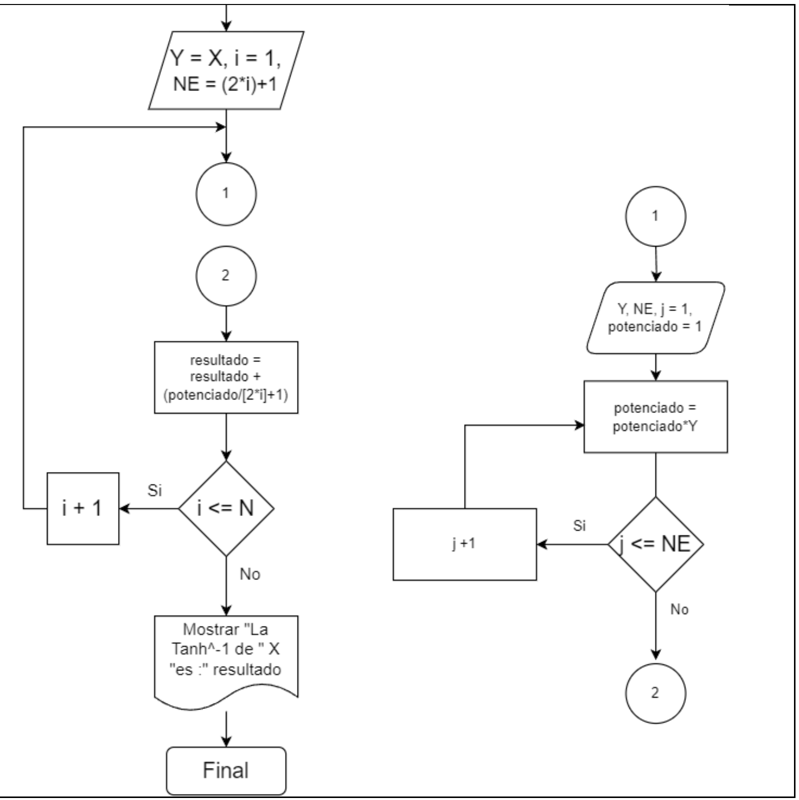
```
Microsoft Windows [Versión 10.0.19045.5555]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Gustavo\Documents\C++\Prog Ing P1>cd "c:\Users\Gustavo\Documents\C++\Prog Ing P1\output"

c:\Users\Gustavo\Documents\C++\Prog Ing P1\output>.\"Tanh-1X.exe"

Tanh^-1(x) Tangente hiperbolica inversa de X

Ingrese un valor de |X| < 1: 0.968

Ingrese el numero de iteraciones con las que desea aproximar Tanh^-1(x): 256

La Tanh^-1(0.96800) = 2.059518585851
c:\Users\Gustavo\Documents\C++\Prog Ing P1\output>
```

Tabla de Contenidos

$Tanh^{-1}(X)$	Tanh^-1(0.968)		Tanh^-1(0.048)		Tanh^-1(-0.329)	
n	Valor	Error	Valor	Error	Valor	Error
2	1.4403299	0.300647293	0.0480369	3.13193E-07	-0.3416413	0.00019069
4	1.6370158	0.205146382	0.0480369	3.13193E-07	-0.3417059	1.6414E-06
8	1.82575408	0.113504442	0.0480369	3.13193E-07	-0.3417064	1.7814E-07
16	1.9674443	0.044706704	0.0480369	3.13193E-07	-0.3417064	1.7814E-07
32	2.0395296	0.00970566	0.0480369	3.13193E-07	-0.3417064	1.7814E-07
64	2.0580837	0.00069671	0.0480369	3.13193E-07	-0.3417064	1.7814E-07
128	2.0595063	5.96615E-06	0.0480369	3.13193E-07	-0.3417064	1.7814E-07
256	2.0595185	4.24401E-08	0.0480369	3.13193E-07	-0.3417064	1.7814E-07

Diagrama	

Inicio
Mostrar "Tanh-1 (X) Tangente hiperbólica de X"
Mostrar "Ingrese un valor de |X| < 1"
X
X >= 1
Si
Mostrar "El valor ingresado no es valido, intente de nuevo"
X = 2
No
X <= -1
Si
No
resultado = X
Mostrar "Ingrese el número de iteraciones/precisión con la que deseas aproximar Tanh-1 (X)"
N
N < 1
Si



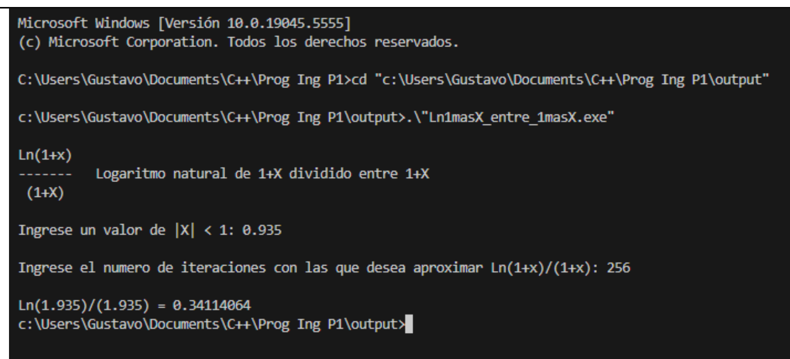
Series Varias

Función	$$\frac{\mathrm{Ln}(1+x)}{1+x}$$
Descripción	El algoritmo aproxima mediante series de Taylor el valor del logaritmo natural de 1 más X para un valor real de X ingresado cuyo valor absoluto sea menor a uno, usando una cantidad N para un valor natural de términos ingresados, devolviendo el valor original de X sumado uno y el resultado de la función con precisión de 12 decimales. Según las pruebas realizadas, el valor idóneo de N iteraciones que maximiza la precisión del algoritmo con respecto al valor teórico ideal depende en gran medida de cada caso, aumentando conforme X se aproxima a uno. La cifra más común en la que los resultados obtienen un error relativo bajo es en torno a 256 términos, lo que se debe considerar al usar este algoritmo.
Código	

	`` /*Aproximar el Logaritmo natural de X mas uno dividido entre X mas uno Ln(X+1) ------- |X|<1 (X+1) Elaborado por Gustavo González De la Cruz*/ #include <stdio.h> /*Se elaboraron funciones usadas frecuentemente con el fin de facilitar la comprensión y depuración del código*/ // Función para calcular X^n double X_a_n(double X, int n) { double resu = 1.0;//Definir un valor por defecto// for (int j = 0; j < n; j++) resu *= X;//Multiplicar X por X N veces// return resu;//Devolver el resultado de X^N// } //Algoritmo principal// int main() { //Declarar variables// double X, res, sig, sum; long int i, j, n=0; printf("\nLn(1+x)"); printf("\n------- Logaritmo natural de 1+X dividido entre 1+X"); //Mostar al usuario la función que se calculará// printf("\n (1+X) \n"); do{ printf("\nIngrese un valor de |X| < 1: ");//Solicitar el valor de X al usuario// scanf("%lf", &X);//Leer el valor de X ingresado// if (X>=1||X<=-1) {//Si el valor absoluto de X es mayor o igual a 1// printf("\nEl valor ingresado no es valido, intente de nuevo");//Indicarle al usuario que el valor no es valido// X = 1.0;//Igualar el valor de X a 1// } } while (X==1.0);//Mientras X sea igual a 1 do{//Solicitar al usuario el numero de iteraciones/precisión con la cual aproximar Ln(1+x)/(1+x)// printf("\nIngrese el numero de iteraciones con las que desea aproximar Ln(1+x)/(1+x): "); scanf("%ld", &n);//Leer el numero de iteraciones a realizar// } while (n < 1); ```

```
    //Calcular Ln(X+1)/(X+1)//
    for ( i = 1; i <= n; i++){
        sig = (i%2)?1:-1;//Definir el signo de la
iteración en base al numero de iteración//
        for ( j = 1, sum = 0; j <= i; j++)
            sum += 1.0/(float)j;//Definir el termino
j-esimo como la sumatoria de uno entre uno hasta el
numero de iteracion//

        res += (X_a_n(X, i))*sum*sig;/*Sumar el
termino i-esimo como la multiplicación del signo
                                    por la sumatoria
de uno entre j, por X elevada a la i-esima potencia*/
    }
    //Mostrar al usuario el valor de X+1 y el
resultado de Ln(X+1)/X+1
    printf("\nLn(%0.3lf)/(%0.3lf) = %0.8lf", 1+X,
1+X, res);
    return 0;
}
```

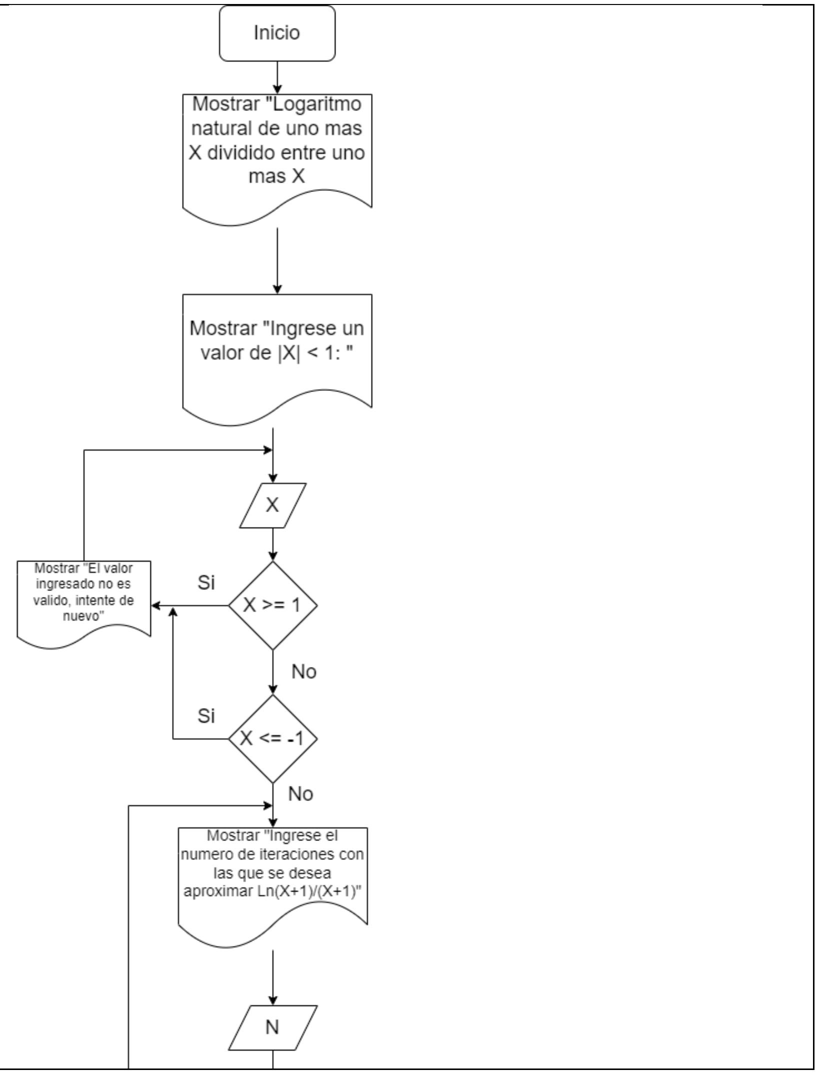
Capturas



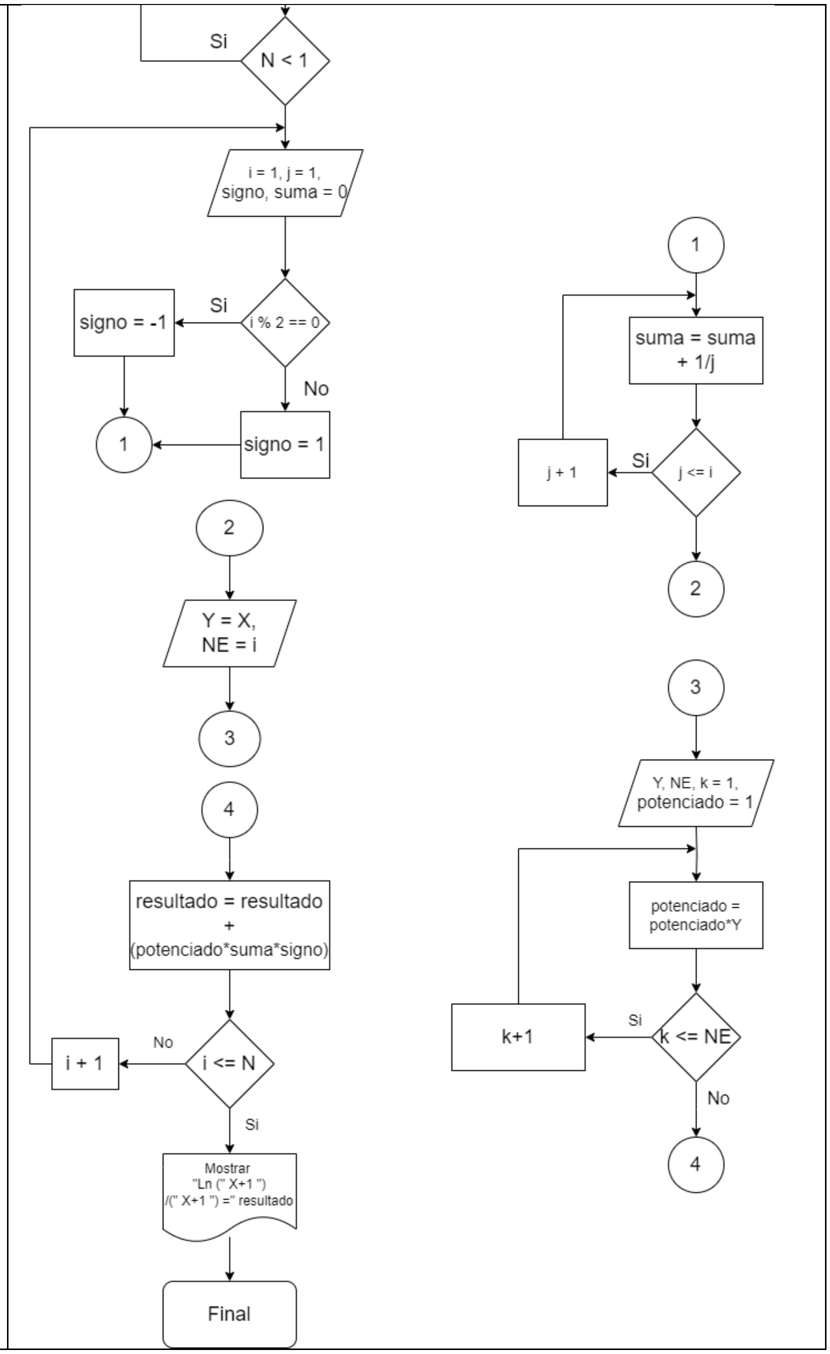
```
Microsoft Windows [Versión 10.0.19045.5555]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Gustavo\Documents\C++\Prog Ing P1>cd "c:\Users\Gustavo\Documents\C++\Prog Ing P1\output"

c:\Users\Gustavo\Documents\C++\Prog Ing P1\output>.\"Ln1masX_entre_1masX.exe"

Ln(1+x)
-------   Logaritmo natural de 1+X dividido entre 1+X
 (1+X)

Ingrese un valor de |X| < 1: 0.935

Ingrese el numero de iteraciones con las que desea aproximar Ln(1+x)/(1+x): 256

Ln(1.935)/(1.935) = 0.34114064
c:\Users\Gustavo\Documents\C++\Prog Ing P1\output>
```

Tabla de Contenidos

$\frac{Ln(1+X)}{1+X}$	Ln(1+0.377) /1+0.377		Ln(1+0.935) /1+0.935		Ln(1+0.79) /1+0.79	
n	Valor	Error	Valor	Error	Valor	Error
2	0.1638065	0.294915724	-0.3763375	2.10317373	-0.14615	1.44933267
4	0.21995663	0.05322462	-0.46999796	2.37772452	-0.05370519	1.16511458
8	0.23200895	0.001346939	-0.44302467	2.2986566	0.13892652	0.57287565
16	0.23232172	6.60524E-07	-0.22136133	1.64888565	0.29057632	0.10663405
32	0.23232187	1.48684E-08	0.11196733	0.67178552	0.32430724	0.0029296
64	0.23232187	1.48684E-08	0.31002779	0.09120267	0.32525953	1.8226E-06
128	0.23232187	1.48684E-08	0.34065836	0.00141401	0.32526012	8.6783E-09
256	0.23232187	1.48684E-08	0.34114064	2.8493E-07	0.32526012	8.6783E-09

Diagrama	

Inicio
Mostrar "Logaritmo natural de uno mas X dividido entre uno mas X
Mostrar "Ingrese un valor de |X| < 1: "
X
Mostrar "El valor ingresado no es valido, intente de nuevo"
Si
X >= 1
No
Si
X <= -1
No
Mostrar "Ingrese el numero de iteraciones con las que se desea aproximar Ln(X+1)/(X+1)"
N



Función	$e^{\mathrm{sen}(x)}$
Descripción	El algoritmo aproxima mediante series de Taylor el valor del número de Euler elevado al seno de X radianes para un valor real de X ingresado, usando una cantidad N para un valor natural de términos ingresados, devolviendo el valor original de X, el

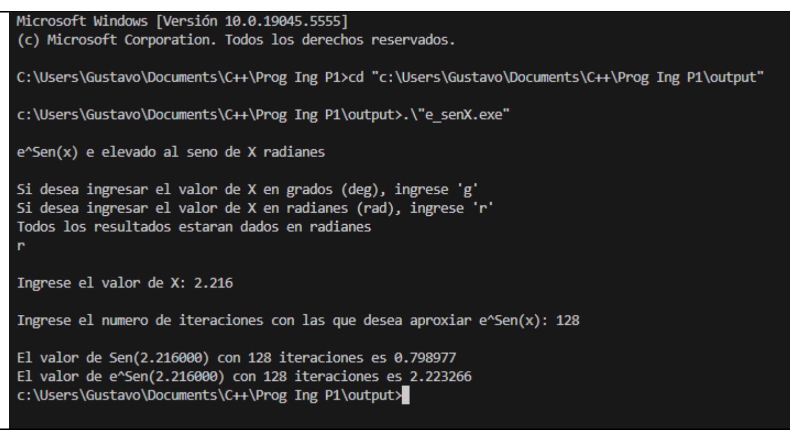
	numero de iteraciones usadas para aproximar la serie y el resultado de la función con precisión de 12 decimales. Dado que la función Seno de X toma entradas y salidas en radianes, se incluyó la posibilidad de elegir si la entrada del ángulo X se realizará en grados o radianes, realizando automáticamente la conversión de unidades partiendo de la elección del usuario. Según las pruebas realizadas, el valor idóneo de N iteraciones que maximiza la precisión del algoritmo con respecto al valor teórico ideal se encuentra en el rango comprendido entre 32 y 128, siendo que usar un numero pequeño de iteraciones genera valores no convergentes con índices de error relativo muy altos, mientas que el uso de valores más cercanos a 256 pueden provocar la indeterminación de la función, en estas pruebas los valores ingresados fueron en radianes como implica la serie original.
Código	(see code below)

```
/*Aproximar e elevado al Seno de X
e^Sen(x)
Elaborado por Gustavo González De la Cruz*/
#include <stdio.h>
/*Se elaboraron funciones usadas frecuentemente con
el fin de
facilitar la comprensión y depuración del código*/

// Función para calcular X^n
double X_a_n(double X, int n) {
    double resu = 1.0;//Definir un resultrado por
defecto//
    for (int j = 0; j < n; j++) {
        resu *= X;//Multiplicar X por X n veces//
    }
    return resu;//Devolver el valor de X^N//
}

// Funcion para calcular el factorial de N
double factor(int N) {
    double resul = 1.0;//Definir un resultado por
defecto//
    for (int k = 2; k <= N; k++) {
        resul *= k;//Multiplicar k N veces//
    }
    return resul;//Devolver el valor de X!//
}
//Algoritmo principal//
int main()
{
    //Declaración de variables//
    char seleccion;
    long int i, j, n=0;
```

```
    double X=0.0, res_s=1.0, res_es=1.0, fact;
    //Se muestra al usuario la función calculada por
el programa//
    printf("\ne^Sen(x) e elevado al seno de X
radianes\n");
    /*Dado que el algoritmo/Serie de Taylor de Sen(x)
requiere el ingreso del valor de X en radianes
    se solicita al usuario que elija el tipo de dato
que desea ingresar*/
    printf("\nSi desea ingresar el valor de X en
grados (deg), ingrese 'g'");
    printf("\nSi desea ingresar el valor de X en
radianes (rad), ingrese 'r'");
    printf("\nTodos los resultados estaran dados en
radianes\n");
    scanf("%c", &seleccion);//Leer la elección del
usuario//
    printf("\nIngrese el valor de X: ");//Solicitar
al usuario que ingrese el valor de X//
    scanf("%lf", &X);//Leer el valor de X ingresado//
    if (seleccion=='g')//En base a la elección del
usuario entre radianes y grados, convertir el ingreso
a radianes//
    {
        //Mostrar al usuario el ingreso en grados
convertido en radianes//
        printf("\nSu ingreso en %lf grados es igual
a:", X);
        X *= 0.0174533;
        printf("\n%lf radianes", X);
    }
    res_s = X;//Asignar al resultado del seno el
valor de X resultante de la conversión, para ahorrar
una iteración//
    do
    {//Solicitar al usuario que ingrese la cantidad
de iteraciones/precisión con la que quiere aproximar
e^Sen(x)//
        printf("\nIngrese el numero de iteraciones
con las que desea aproxiar e^Sen(x): ");
        scanf("%ld", &n);
    } while (n<1);//Hasta que ingrese un valor mayor
a 1//
    for ( i = 1; i < n; i++)//Calcular el valor de
Sen(x)//
        /*Definir el signo de la iteración en base al
numero de iteración,
        multiplicarlo por el valor de X elevado al
doble del numero de iteración mas uno,
        y dividirlo entre el factorial del doble del
numero de iteración mas uno*/
```

```
        res_s += ((i%2)?-1:1)*(X_a_n(X,
(2*i)+1))/factor((2*i)+1);//Sumar el resultado de la
iteración al resultado de Sen(X)//
    //Imprimir el valor de X, el numero de
iteraciones usadas para aproximarlo y el resultado
del Sen(X)//
    printf("\nEl valor de Sen(%lf) con %ld
iteraciones es %lf", X, i, res_s);

     // Calcula la serie para e^Y donde Y es igual al
Sen(x)
     for (j = 1; j <= n; j++) {
        fact = 1; // Reinicia el valor del termino
        fact = X_a_n(res_s, j)/factor(j);/*Definir el
valor del termino como el Sen(X) elevado al numero de
iteración
                                        dividido
entre el factorial del numero de iteración*/
        res_es += fact; // Suma el término calculado
a e^x
    }
    /*Imprimir al usuario el Valor de X ingresado, el
numero de iteraciones realizadas en cada proceso y el
valor
    aproximado resultante*/
    printf("\nEl valor de e^Sen(%lf) con %ld
iteraciones es %lf", X, i, res_es);
    return 0;
}
```

Capturas

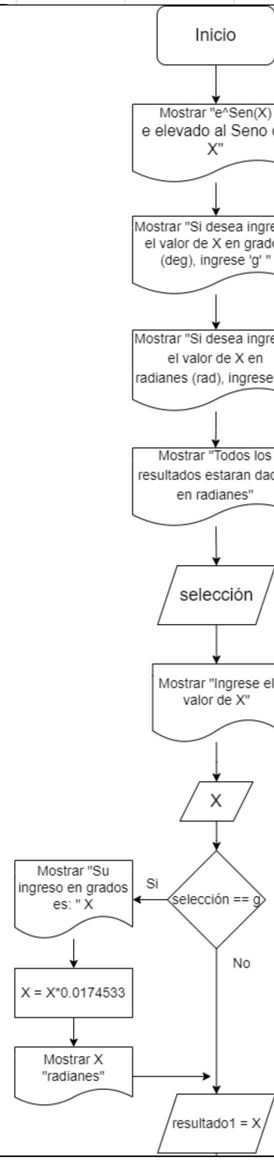


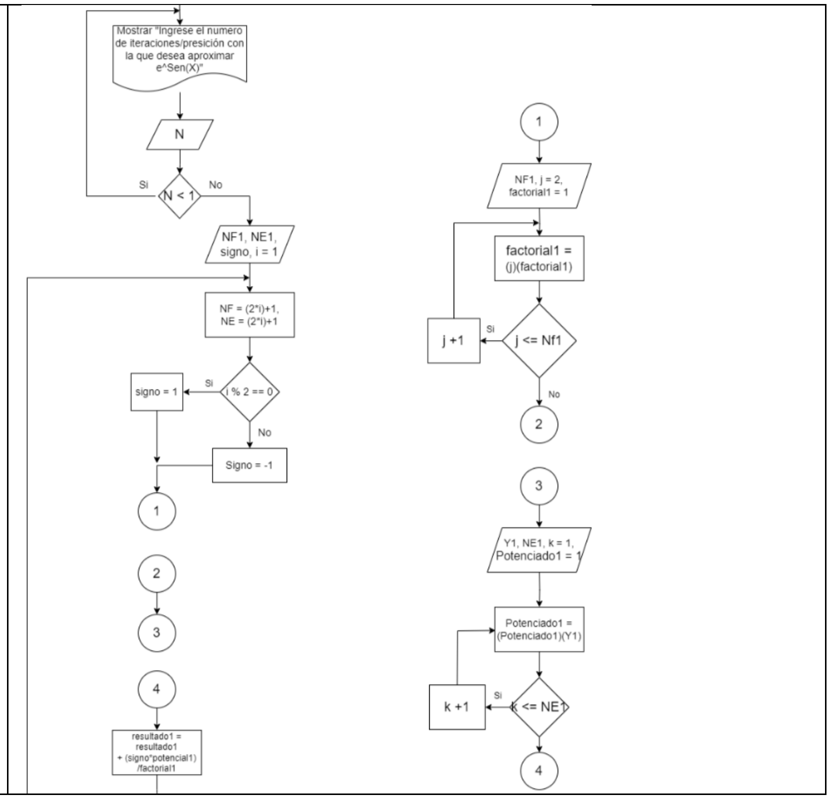
```
Microsoft Windows [Versión 10.0.19045.5555]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Gustavo\Documents\C++\Prog Ing P1>cd "c:\Users\Gustavo\Documents\C++\Prog Ing P1\output"

c:\Users\Gustavo\Documents\C++\Prog Ing P1\output>.\"e_senX.exe"

e^Sen(x) e elevado al seno de X radianes

Si desea ingresar el valor de X en grados (deg), ingrese 'g'
Si desea ingresar el valor de X en radianes (rad), ingrese 'r'
Todos los resultados estaran dados en radianes
r

Ingrese el valor de X: 2.216

Ingrese el numero de iteraciones con las que desea aproxiar e^Sen(x): 128

El valor de Sen(2.216000) con 128 iteraciones es 0.798977
El valor de e^Sen(2.216000) con 128 iteraciones es 2.223266
c:\Users\Gustavo\Documents\C++\Prog Ing P1\output>
```

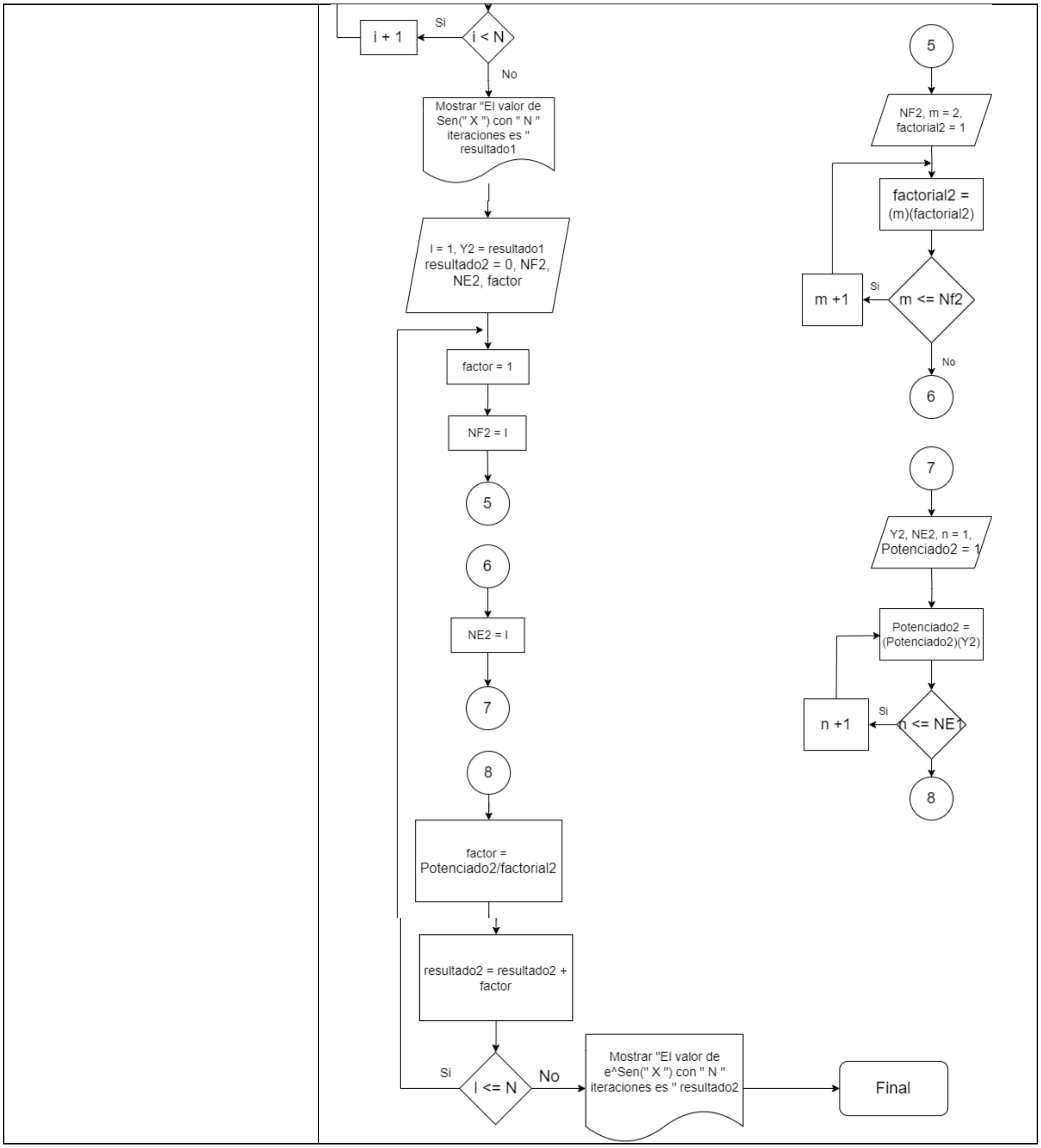
Tabla de Contenidos

$e^{Sen(X)}$	e^Sen(2.216)		e^Sen(3.504)		e^Sen(9.704)	
n	Valor	Error	Valor	Error	Valor	Error
2	1.483266	0.332843674	4.054741	4.780054155	10025.27368	13205.6062
4	2.212673	0.004764628	0.576593	0.178063712	47313440448	6.2327E+10
8	2.223266	1.64132E-08	0.701502	5.29021E-06	2.05942E+12	2.7129E+12
16	2.223266	1.64132E-08	0.701506	4.11812E-07	0.75908	4.0203E-05
32	2.223266	1.64132E-08	0.701506	4.11812E-07	0.759111	6.3436E-07
64	2.223266	1.64132E-08	0.701506	4.11812E-07	0.759111	6.3436E-07
128	2.223266	1.64132E-08	0.701506	4.11812E-07	0.759111	6.3436E-07
256	2.223266	1.64132E-08	0.701506	4.11812E-07	-1.#IND0000	1

Diagrama




Mostrar "Ingrese el numero de iteraciones/presición con la que desea aproximar e^Sen(X)"
N
Si
N < 1
No
NF1, NE1, signo, i = 1
NF = (2*i)+1, NE = (2*i)+1
Si
signo = 1
i % 2 == 0
No
Signo = -1
1
2
3
4
resultado1 = resultado1 + (signo*potencial1) /factorial1
1
NF1, j = 2, factorial1 = 1
factorial1 = (j)(factorial1)
Si
j +1
j <= Nf1
No
2
3
Y1, NE1, k = 1, Potenciado1 = 1
Potenciado1 = (Potenciado1)(Y1)
Si
k +1
k <= NE1
4



Conclusión

Tras haber desarrollado una serie de algoritmos destinados a resolver la problemática de encontrar un valor relativamente corto o exacto del valor real, se identificaron diversas complicaciones durante el proceso. Una de las principales dificultades fue el manejo de decimales, ya que, para lograr resultados precisos, era necesario considerar la mayoría de estos valores. Esto implicaba un mayor cuidado en las operaciones y cálculos intermedios, ya que incluso un pequeño error en la aproximación decimal podía afectar significativamente el resultado final. Además, se evidenció que la precisión en los cálculos estaba estrechamente relacionada con la cantidad de términos empleados en las series, lo que llevó a una reflexión profunda sobre cómo optimizar los algoritmos sin comprometer la exactitud de los resultados.

Por otro lado, el uso de ciclos en la ejecución de los algoritmos presentó ciertos retos, especialmente en los casos de compilación, donde se obtenían resultados menores o mayores a los esperados. Este tipo de errores no siempre se solucionaban simplemente reiniciando el compilador, sino que requerían un análisis detallado para identificar la causa del fallo y corregirlo adecuadamente. En algunos casos, fue necesario depurar cada segmento de código y revisar la lógica implementada, lo cual fortaleció nuestras habilidades analíticas y de resolución de problemas. Estas dificultades, aunque desafiantes, ofrecieron la oportunidad de aprender a enfrentar situaciones complejas y a mantener la paciencia y la perseverancia frente a los obstáculos.

Asimismo, la implementación de las series matemáticas representó un desafío significativo. La complejidad del concepto generó incertidumbre durante la etapa de compilación y análisis de resultados, especialmente en la interpretación y validación de los datos obtenidos. Sin embargo, enfrentar esta dificultad resultó enriquecedor, ya que se fortaleció la capacidad de resolución de problemas, una habilidad fundamental en nuestra carrera. De hecho, fue posible obtener los primeros programas funcionales en tan solo un día de trabajo, mientras que los más complejos requirieron horas adicionales de dedicación y análisis. Este proceso evidenció la importancia de la práctica constante y del aprendizaje continuo, elementos clave para el desarrollo de habilidades técnicas avanzadas.

Una observación relevante durante el proceso fue que, al calcular el valor de utilizando potencias pares, se observó que al incrementar el número de términos a 16, el margen de error se reducía prácticamente a 0%. Este hallazgo no solo evidenció un desarrollo positivo en la resolución del problema, sino que también subrayó la importancia de ajustar los parámetros del algoritmo para optimizar su precisión. Además, se destacó la necesidad de comprender cómo el comportamiento de las series influye en la aproximación final, lo que abre la puerta a futuras investigaciones y mejoras en el diseño de algoritmos matemáticos más eficientes.

En conclusión, la experiencia adquirida a lo largo de este proceso no solo permitió afianzar conocimientos técnicos, sino que también fortaleció la capacidad de análisis y adaptación frente a desafíos complejos, lo cual es esencial para el desarrollo profesional en el ámbito de la ingeniería. Cada obstáculo superado contribuyó a forjar una mentalidad más crítica y reflexiva, promoviendo un enfoque más riguroso y meticuloso en la resolución de problemas. Este aprendizaje práctico se convierte, por tanto, en un recurso invaluable para enfrentar futuros desafíos en el ámbito académico y profesional.