

Universidad Mariano Gálvez de Guatemala

Facultad de Ingeniería en Sistemas de la Información y la Computación

Campus Villa Nueva, Guatemala

Ingeniería en Sistemas -5090

Curso: PROGRAMACIÓN I

Licenciado/Ingeniero titular: Ing. Carlos Alejandro Arias

Depuración y manejo de excepciones en C++ Lab#9

ESTUDIANTES: Carlos Eduardo García Cortez

CARNET: 5090-24-14824

FECHA: 24/03/2025

Parte 1: Teoría del laboratorio

- ✓ Utilizar breakpoints (puntos de quiebre) para pausar la ejecución del programa.
- ✓ Monitorear variables con la herramienta Watch en Visual Studio Community.
- ✓ Manejar excepciones en C++ utilizando bloques try, throw y catch.
- ✓ Comprender el comportamiento de la recursividad a través de un ejemplo práctico.
- ✓ Usar herramientas avanzadas de depuración como Call Stack y Conditional Breakpoints.

Código base

```
#include <iostream>
```

```
using namespace std;
```

```
// Función recursiva para calcular el factorial
```

```
int factorial(int n) {
```

```
    if (n < 0) {
```

```
        throw invalid_argument("Error: Factorial de un número negativo no está definido.");
```

```
    }
```

```
    if (n == 0 || n == 1) {
```

```
        return 1;
```

```
    }
```

```
    return n * factorial(n
```

```
        -
```

```
        1);
```

```
    }
```

```
int main() {
```

```
    try {
```

```
        int numero;
```

```
        cout << "Introduce  
un número para calcular el factorial: ";
```

```
        cin >> numero;
```

```
        int resultado = factorial(numero);
```

```
        cout << "El factorial de " << numero << " es: " << resultado << endl;
```

```

} catch (const exception &e) {

cerr << "Excepción capturad
a: " << e.what() << endl;

}

return 0;

}

```

Parte 2: Depuración del programa

Agregar un breakpoint

Haz clic en el margen izquierdo junto a esta línea, tendrás que ver que un punto rojo está indicando que el breakpoint está activo:

int resultado = factorial(numero);



```

        return 1;
    }
    return n * factorial(n - 1);
}

int fibonacci(int n) {
    if (n < 0) {
        throw invalid_argument("Error: No existe Fibonacci de un número negativo.");
    }
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    try {
        int numero;
        cout << "Introduce un número para calcular su factorial: ";
        cin >> numero;

        int resultadoFact = factorial(numero);
        cout << "El factorial de " << numero << " es: " << resultadoFact << endl;

        cout << "Introduce un número para calcular Fibonacci: ";
        cin >> numero;

        int resultadoFib = fibonacci(numero);
        cout << "El número de Fibonacci " << numero << " es: " << resultadoFib << endl;
    }
}






```

Ejecutar el programa en modo depuración

Presiona F5 o selecciona iniciar depuración

Introduce un número cuando se te pida.

El programa debe detenerse en el breakpoint.

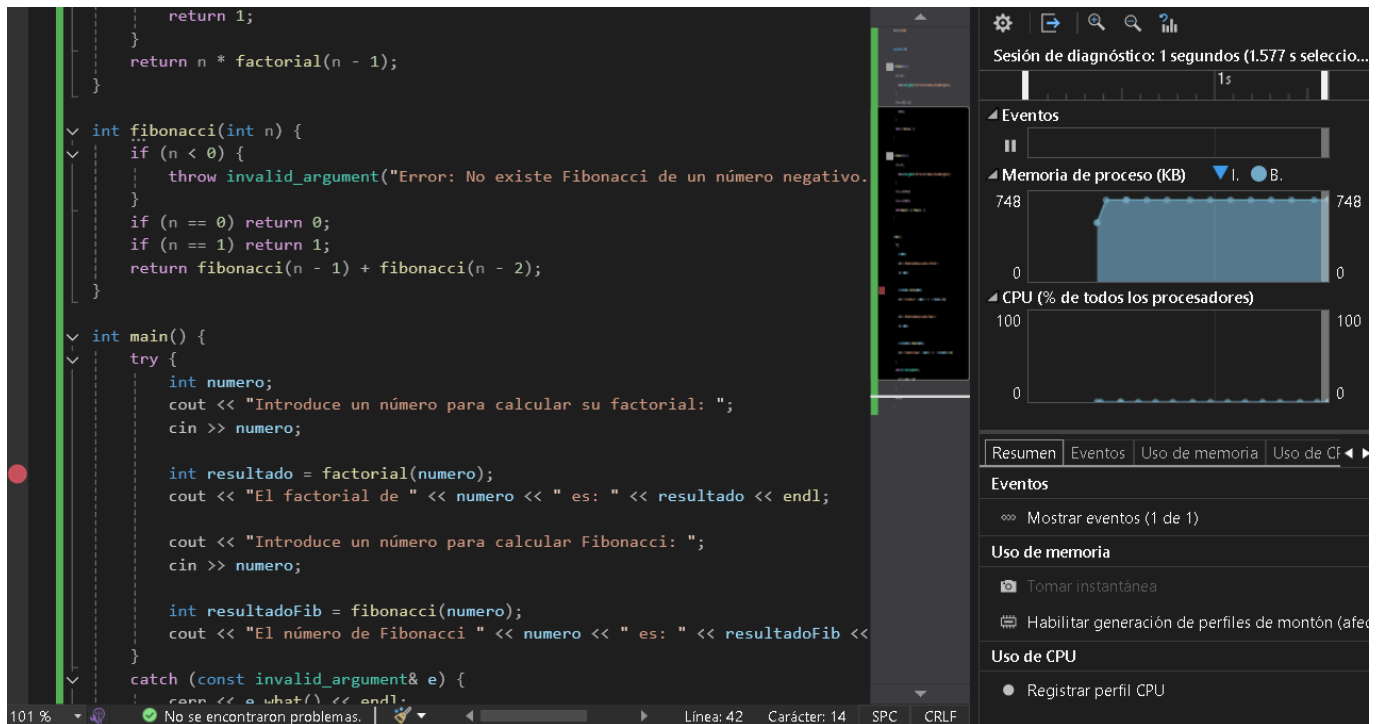
Automático		
Buscar (Ctrl+E)  ← → Profundidad de búsqueda: 3  		
Nombre	Valor	Tipo
 numero	5	int
 resultadoFact	-858993460	int

Inspeccionar variables (watch)

Ve al menú Depurar -> Ventanas -> Inspección (Watch)

Agrega las variables número y resultado.

Observa cómo cambian sus valores paso a paso presionando F10.



```

return 1;
}
return n * factorial(n - 1);
}

int fibonacci(int n) {
    if (n < 0) {
        throw invalid_argument("Error: No existe Fibonacci de un número negativo.");
    }
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    try {
        int numero;
        cout << "Introduce un número para calcular su factorial: ";
        cin >> numero;

        int resultado = factorial(numero);
        cout << "El factorial de " << numero << " es: " << resultado << endl;

        cout << "Introduce un número para calcular Fibonacci: ";
        cin >> numero;

        int resultadoFib = fibonacci(numero);
        cout << "El número de Fibonacci " << numero << " es: " << resultadoFib << endl;
    }
    catch (const invalid_argument& e) {
        cerr << e.what() << endl;
    }
}

```

Manejo de excepciones

Ejecuta el programa con F5 e introduce un número negativo.

Observa cómo el programa lanza una excepción y se muestra el mensaje del catch.

```

Introduce un número para calcular su factorial: -3
Error: No existe factorial de un número negativo.

```

Modificar el manejo de excepciones

Añade un catch genérico después del existente.

```
catch (...) {
```

```
    cerr << "Se ha producido un error desconocido." << endl;
```

```
}
```

Analizar la recursividad

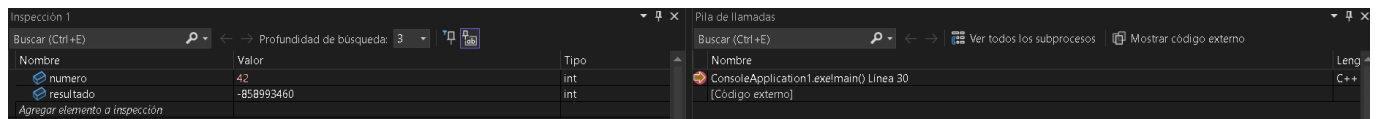
Agrega otro breakpoint en la línea

```
return n * factorial(n - 1);
```

Ejecuta el programa y utiliza F11 para entrar en la función recursiva.

Explorar la pila de llamadas (Call stack)

Observa cómo se apilan las llamadas recursiva hasta alcanzar el caso base.



Desafío adicional

Agregar una nueva función recursiva:

Modificar el programa para calcular n-ésimo número de Fibonacci

```
int fibonacci(int n) {  
  
    if (n < 0) {  
        throw invalid_argument("Error: No existe Fibonacci de un número negativo.");  
    }  
    if (n == 0) return 0;  
    if (n == 1) return 1;  
    return fibonacci(n - 1) + fibonacci(n - 2);  
}
```

Ajusta el código para invocar esta función y realiza el mismo análisis con breakpoint.

```

#include <iostream>

using namespace std;

int factorial(int n) {
    if (n < 0) {
        throw invalid_argument("Error: No existe factorial de un número negativo.");
    }
    if (n == 0 || n == 1) {
        return 1;
    }
    return n * factorial(n - 1);
}

int fibonacci(int n) {
    if (n < 0) {
        throw invalid_argument("Error: No existe Fibonacci de un número negativo.");
    }
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    try {
        int numero;
        cout << "Introduce un número para calcular su factorial: ";
        cin >> numero;

        int resultado = factorial(numero); // 2,751 ms transcurridos
        cout << "El factorial de " << numero << " es: " << resultado << endl;

        cout << "Introduce un número para calcular Fibonacci: ";
        cin >> numero;

        int resultadoFib = fibonacci(numero);
        cout << "El número de Fibonacci " << numero << " es: " << resultadoFib << endl;
    }
    catch (const invalid_argument& e) {
        cerr << e.what() << endl;
    }
    catch (...) {
        cerr << "Se ha producido un error desconocido." << endl;
    }
    return 0;
}

```

Parte 3: Evaluación

¿Cuál es la diferencia entre F10 y F11?

F10 ejecuta la línea actual sin entrar en las funciones que llama, y F11 entra dentro de las funciones llamadas en la línea actual.

¿Por qué es importante el manejo de excepciones en un programa?

Permite controlar errores inesperados de forma estructurada, además de que evita que el programa termine abruptamente.

¿Cómo se comporta la pila de llamadas con funciones recursivas?

Cada llamada recursiva añade un nuevo marco, y la pila crece hasta alcanzar el caso base.