

UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE

Departamento de Ciencias de la Computación

Estructura de Datos

**Integrantes:**

* Díaz Stefany
* Tipán Reishel
* Viscaino Reishel

**NRC:** 14542

**Fecha:** 22/05/2024

**1. Introducción**

En este informe se presentan tres programas implementados en C + + que abordan diferentes conceptos clave en la programación moderna, el primer programa demuestra el uso de funciones lambda para realizar el ordenamiento de un conjunto de datos, el segundo programa implementa la serie de Fibonacci utilizando una técnica avanzada de recursión conocida como el punto fijo y finalmente, el tercer programa aborda la gestión de memoria estática y dinámica, un aspecto fundamental en el desarrollo eficiente y seguro de aplicaciones en C + +.

**2. Desarrollo**

**2. 1. Programa: Ordenamiento y Lambda**

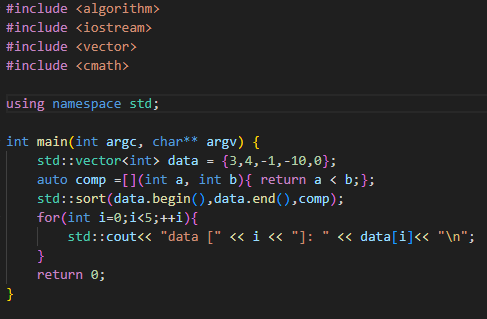
**Descripción**

Este primer programa tiene como objetivo ordenar un conjunto de enteros utilizando el algoritmo de ordenamiento proporcionado por la biblioteca estándar de C + +. Para esto, se hace uso de una función lambda que define la lógica de comparación entre los elementos.

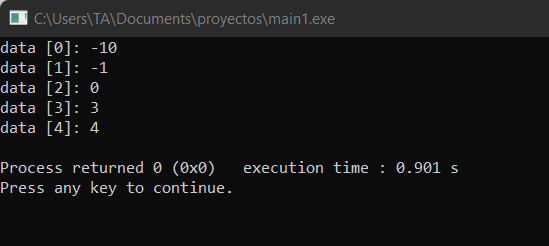
**Explicación del Código**

Se define un vector de enteros y una función lambda que compara dos enteros. Luego, se utiliza `std::sort` para ordenar ascendentemente el vector basado en esta lambda. Finalmente, se imprimen los elementos ordenados.

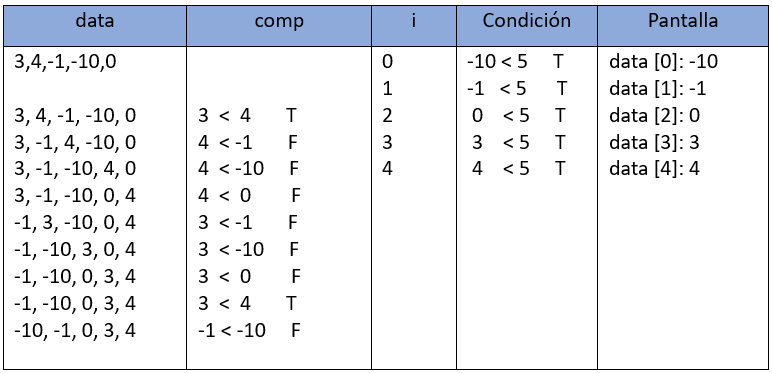
Función Lambda: *auto comp = [](int a, int b) { return a < b; };*

****

**Ejecución del programa**



**Matriz de pruebas de escritorios**



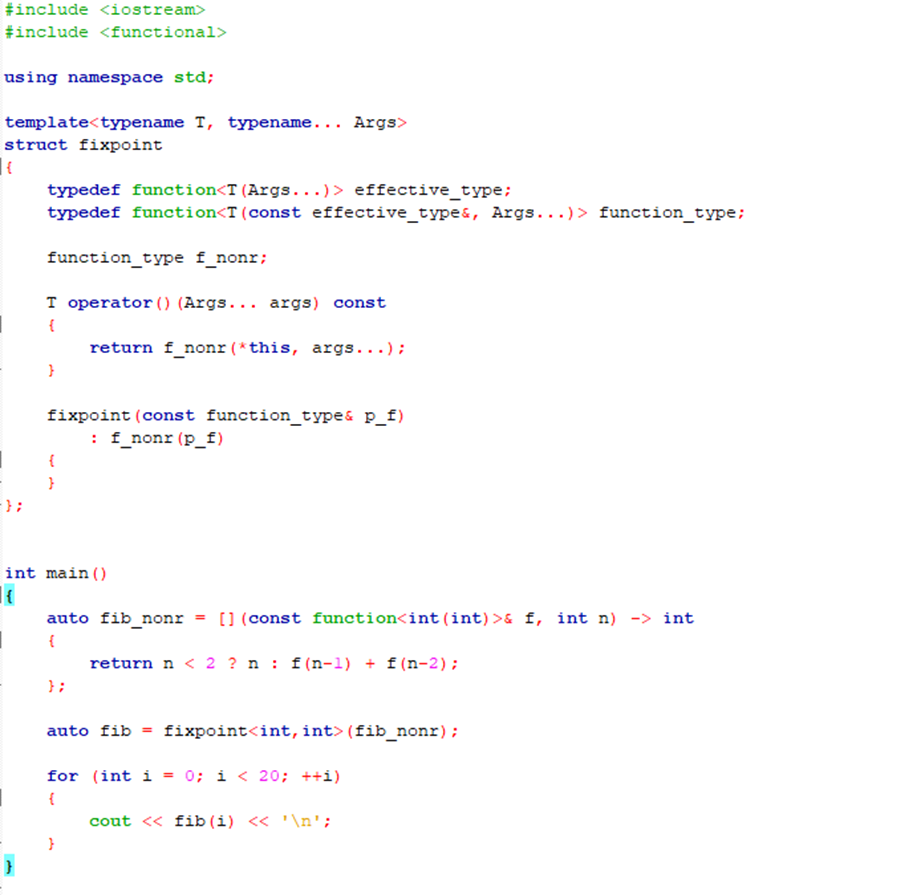
**2. 2 Programa: Recursión con Punto Fijo (Fixpoint)**

**Descripción**

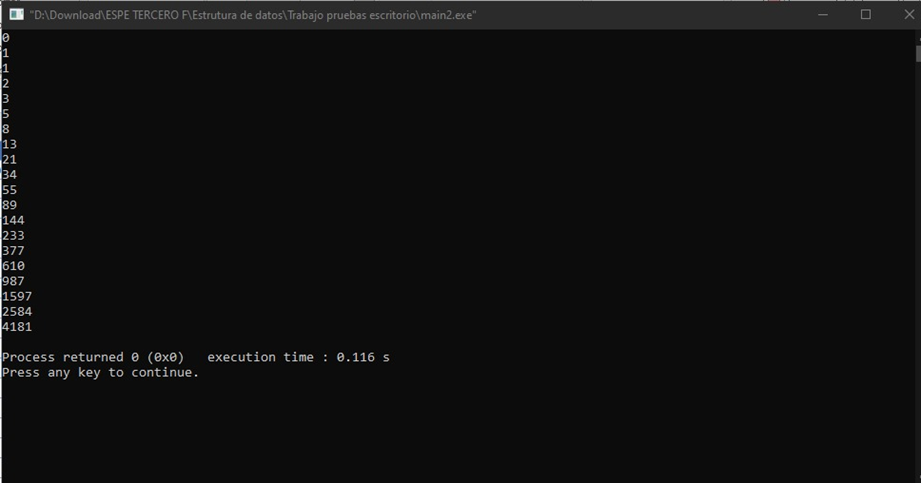
El segundo programa demuestra una técnica avanzada de recursión mediante la implementación de la serie de Fibonacci. Se utiliza una estructura `fixpoint` para manejar la recursión de manera más flexible y genérica.

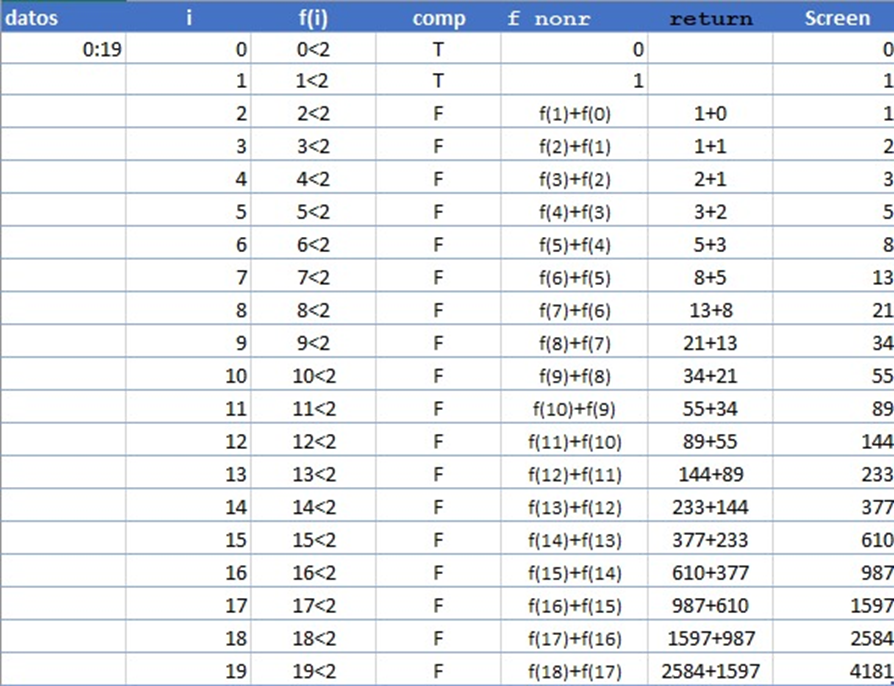
**Explicación del Código**

Se identificó un programa que aplica de manera recursiva métodos para obtener los elementos de la serie Fibonacci. El programa crea un template(plantilla) la cual almacena los datos generados en la función main, para consiguiente aplicar de manera recursiva la función **fib\_nor** que realiza la operación matemática de suma de los valores obtenidos previamente en la serie.  
Compara los valores generados por el bucle for es decir el valor de “i” con la condición n<2  
para así imprimir los datos, si la condición no se cumple se “activa” el llamamiento a la función recursiva, lo cual determina que datos ingresaran en la función **fib\_nor** e imprime dicho resultado.

****

**Ejecución del programa**



**Matriz de pruebas de escritorios**

**2.3. Programa: Gestión de Memoria Estática y Dinámica**

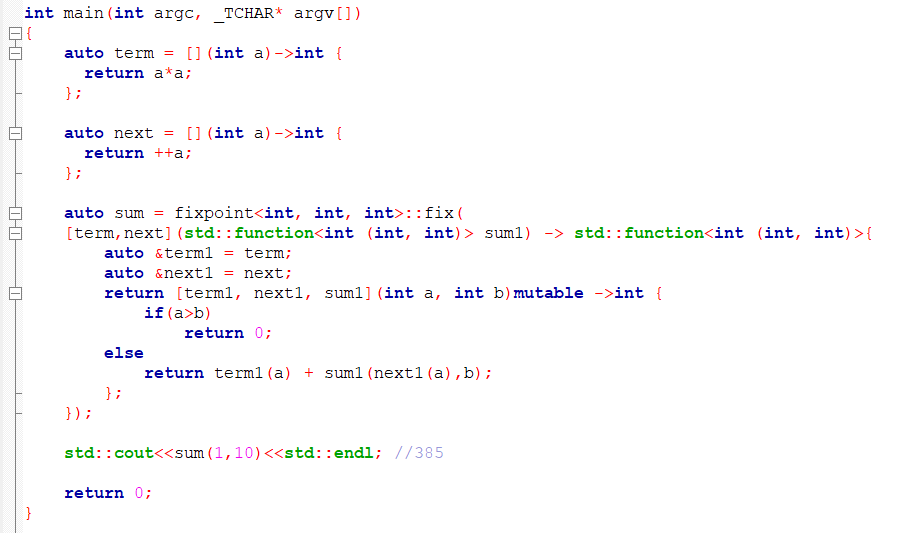
**Descripción**

El tercer programa permite calcular los valores de una serie (desarrollada por el programador), en base a el uso de la recursividad usando fixpoint para crear funciones recursivas sin la necesidad de referenciarse directamente.

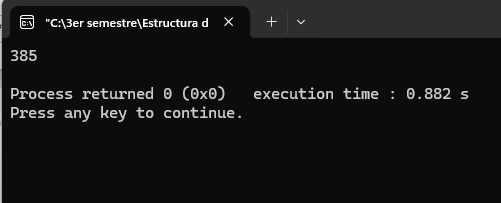
**Explicación del Código**

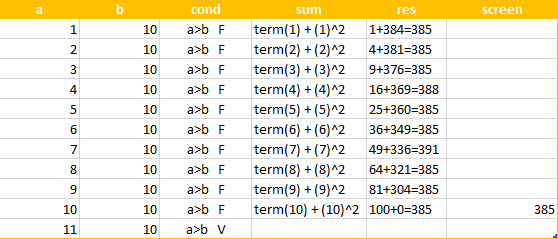
**Template fixpoint:** Define un mecanismo para implementar recursión usando lambdas y funciones.

* *‘fix’* es un método estático que retorna una función que aplica el punto fijo (Y combinator) para realizar la recursión.
* *‘term’* es un lambda que calcula el cuadrado de un número y *‘next’* igualmente es un lambda que calcula el incremento de un número.
* *‘sum’* usa *‘fixpoint'* para crear una función recursiva que suma los cuadrados de los números en un rango.



**Ejecución del programa**

****

**Matriz de pruebas de escritorios**

**3. Conclusiones**

En el segundo ejercicio se define una plantilla *`fixpoint`* que permite encapsular la lógica de recursión. La lambda *`fib\_nonr`* implementa la lógica no recursiva directa de Fibonacci. Se crea una instancia de *`fixpoint`* y se utiliza para calcular e imprimir los primeros 20 números de Fibonacci.

La estructura fixpoint definida en el código permite que una función lambda se llama recursivamente sin necesidad de nombrarse a sí misma, sin embargo, como programador es más compleja esta funcionalidad y genera sobrecarga innecesaria si la recursión se la podría hacer directamente. En el caso del tercer ejercicio tenemos la facilidad de simplificar el código pero debido a la utilidad de la recursividad es importante declararlo para sus próximos usos.