Alejandro Ruiz López

**Actividad individual 1 - Unidades 1 y 2**

**Github**: https://github.com/AleRui/Estructuras_Lineales

PROGRAMACIÓN CON ESTRUCTURAS LINEALES (2024PAGO001202M2100)

## Librerias:

- `#include <algorithm>` -> contiene algortimos de búsqueda, partición, ordenación, etc
- `#include <iostream>` -> Para imprimir con cout punteros por ejemplo
- `#include <fstream>` -> flujos i/o a ficheros
- `#include <print>` -> contiene las funciones std::print y std::println (incluye salto de línea)
- `#include <ranges>` -> funciones para trabajar con rangos, algo iteradod e princpio a fin, -> contiene vistas, visita por conjuntos
- `#include <string>` -> cadena de caracteres estándar
- `#include <vector>` -> contenedor secuencial (contiguos en memoria vistual) recomendado por defecto en C++
- `#include <typeinfo>` -> Library tipo de variable.
- `#include <generator>` // Library Generador.
-

Mapa conceptual Unidades 1 y 2 (Programación Genérica)

| Liberia | Uso | Detalles |
|---|---|---|
| `* #include <print>` | `std::println` | Librería para imprimir línea |
| `* #include <vector>` | `std::vector<std::string>{};` | Librería para usar vectores, hay que especificar de que clase es el vector. Formateadores :> izq :< der :^center |
| | `vector.push_back(entidad);` | Función para introducir en vector al final. |
| `#include <fstream>` | `std::ifstream{"../../path/name_file.jsonl", std::ios::binary};` | Función para introducir en vector al final. |
| | `struct NombreStruct { ... };` | Clase con todos los elementos públicos por defecto. |
| `#include <nlohmann/json.hpp>` | `NLOHMANN_DEFINE_TYPE_NON_INTRUSIVE (Class, param1, param2);` | Macro (Se deben evitar), informo a la biblioteca de la clase y en el orden. |
| `#include <nlohmann/json.hpp>` | `nlohmann::json::parse(ln).get<ClassName>();` | Parsea líneas de un archivo jsonl como objetos de una clase dada. |
| `std::views` | `stdv::chunk_by(indicacion)` | Crea vistas; Agrupa valores de un vector por el tipo de parametro. |
| `std::ranges;` | `stdr::begin(chunked_vector_chunk)->parametro;` | Va al primer elemento del vector y coge el tipo del primer parámetro. |
| `std::views` | `stdv::chunk_by(indicacion)` | Crea vistas; Agrupa valores de un vector por el tipo de parametro. |
| `#include <ranges>` | `stdr::count_if(vector_vista_chunk, &ClassName::parametro);` | Contador de elementos de una vista. |
| `#include <ranges>` | `stdr::distance(target_chunk);` | Contador de elementos |

| Liberia | Uso | Detalles |
|---|---|---|
| | | de un vector. |
| `#include <ranges>` | `stdr::count_if(vector_vista_chunk, &ClassName::parametro);` | Contador de elementos de una vista. |
| | `std::unordered_map<std::string, int> contador;` | Unordered_map contiene elementos solo en forma de pares (clave-valor). |
| | std::generator | Generador |
| | template | Plantilla que convierte en familia uniparamétrica de funciones, clases. |
| | | |

## Punteros:

```
auto n = int{0}; // ocupa 4 bytes
// Buscar dirección de memoria virtual de esta variable
// y quiero guardar esa dirección en memoria en otra variable
int* p1 = &n; // & es un operador unario para obtener la dirección del entero "n"
              // int* p es un puntero, dirección en memoria de un objeto
auto d = double {9.9}; // ocupa 8 bytes
double* p2 = &d;       // Un puntero en memoria ocupa
// std::println no puede imprimir puntero como tales.
std::cout << p1 << '\n';
std::cout << p2 << '\n';

// cuanto ocupa en memoria p1 y p2
// todos ocupan 8 bytes porque almacenan direcciones de memoria
std::println("{} bytes, {} bytes", sizeof(p1), sizeof(p2));
```