UCSM Esc. Prof. de Ingeniería de Sistemas

INFORME DE PRÁCTICAS

Marzo - 2025

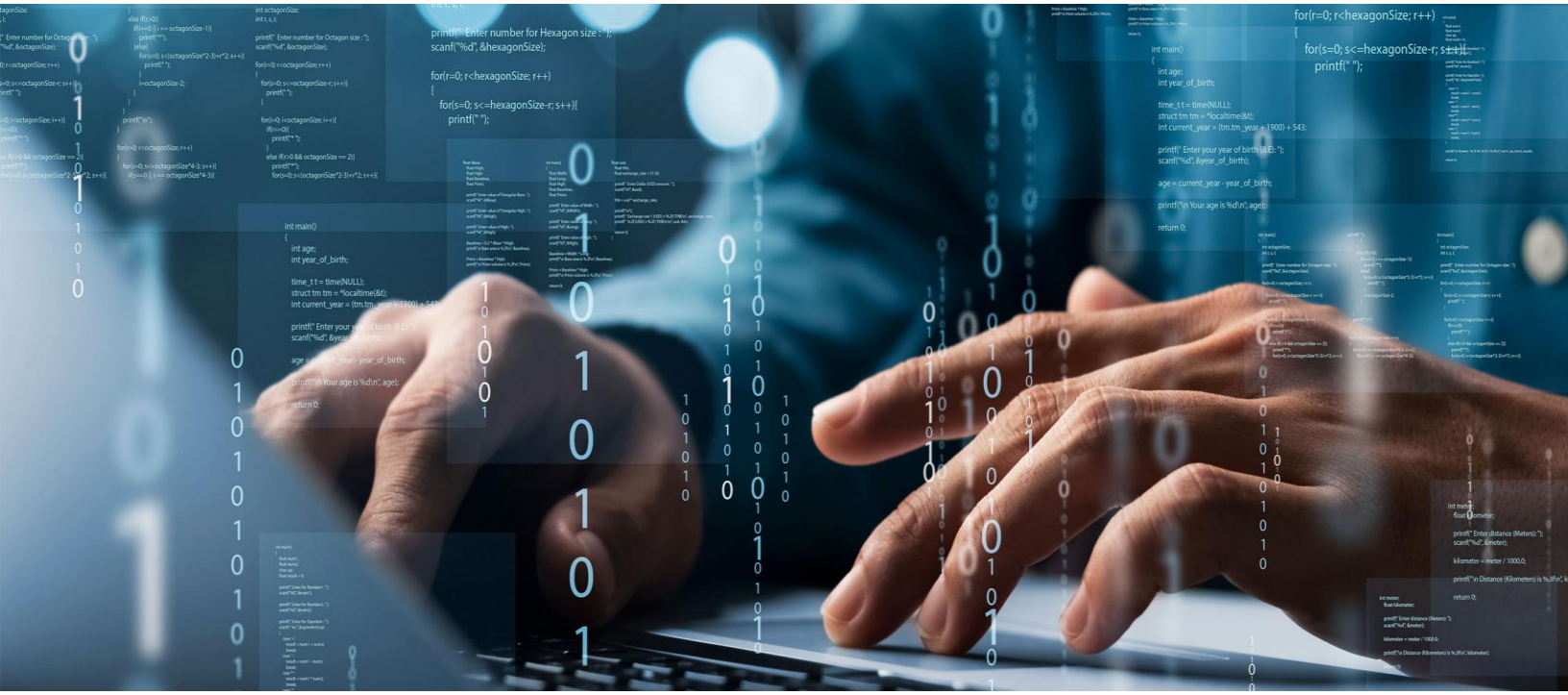
LENGUAJES DE PROGRAMACIÓN II

Práctica N° 12: Standard Template Library en C++

Elaborado por:

HUANQUI LUQUE PIEROL YARENAPELLIDOS

SANCHEZ YDME RODRIGO FABRIZIO



GRUPO N° 0X

PRÁCTICAS DE SISTEMAS INTELIGENTES

Presentado por:

2024002019	HUANQUI LUQUE PIEROL YAREN	100%
2024002172	SANCHEZ YDME RODRIGO FABRIZIO	100%

RECONOCIMIENTOS

Se reconoce la importancia de la STL como una herramienta poderosa que simplifica y optimiza el proceso de desarrollo en C++. La comprensión y correcta aplicación de sus componentes contribuyen a la creación de programas eficientes, reutilizables y fáciles de mantener. La formación en esta biblioteca es esencial para resolver problemas de programación complejos y para potenciar las habilidades de los futuros ingenieros de sistemas.

PALABRAS CLAVES

STL, Contenedores, Algoritmos, Iteradores, Programación en C++, Eficiencia, Reutilización, Datos, Estructuras de Datos, Programación Orientada a Datos.

ÍNDICE

ÍNDICE DE TABLAS Y FIGURAS

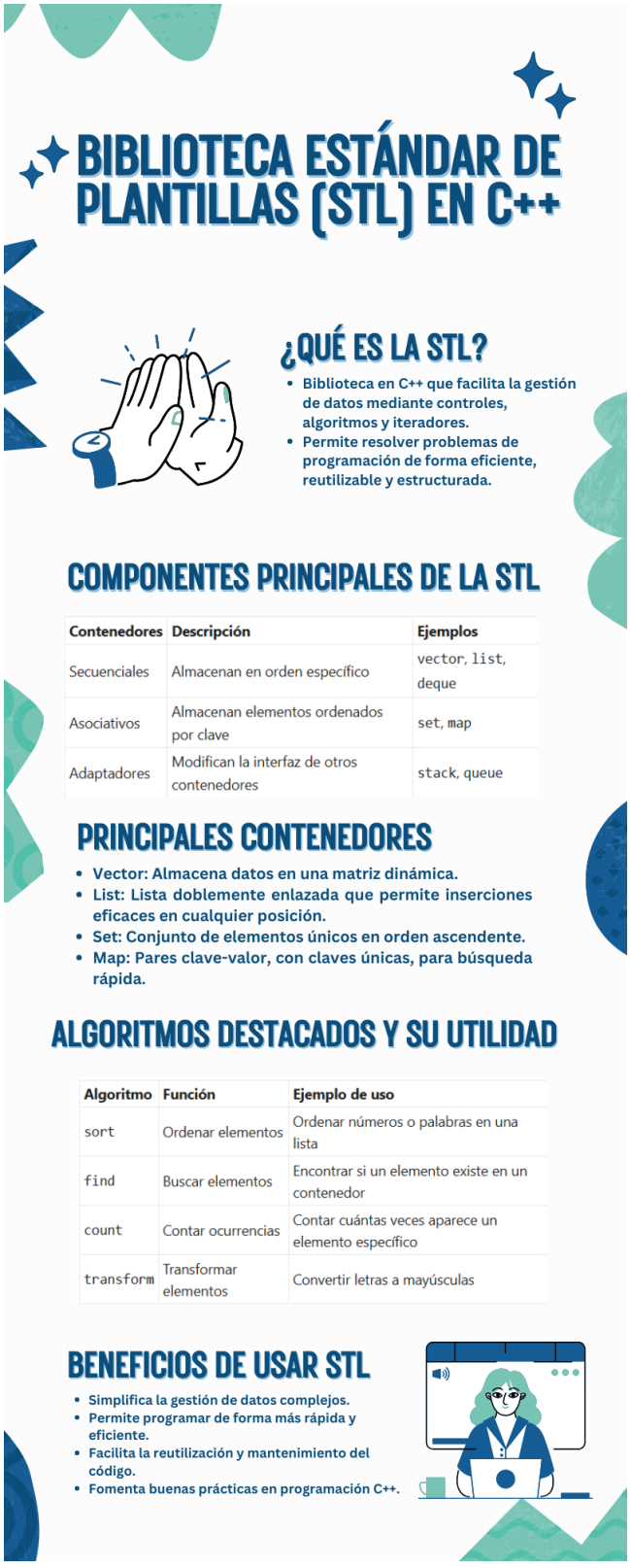
1. RESUMEN

La práctica de la Sesión 12 de Programación II se centra en la utilización de la Biblioteca Estándar de Plantillas (STL) en C++, abordando sus componentes principales: contenedores, algoritmos e iteradores. A través de ejemplos y ejercicios prácticos, los estudiantes aprenden a manipular datos de forma eficiente y a resolver problemas comunes de programación utilizando las estructuras y funciones predefinidas de la STL. La sesión busca fortalecer las habilidades en programación eficiente, facilitando el desarrollo de soluciones reutilizables y optimizadas en C++

2. INTRODUCCIÓN

La STL en C++ es una biblioteca fundamental que proporciona agrupar estructuras de datos y algoritmos, permitiendo a los programadores gestionar y manipular datos de manera efectiva. En la presente sesión, se explorarán los diferentes tipos de contenedores, algoritmos y el uso de iteradores, facilitando la comprensión y aplicación práctica de estas herramientas. La inversión en el aprendizaje de la STL permite mejorar la productividad, garantizar código más eficiente y promover buenas prácticas en programación.

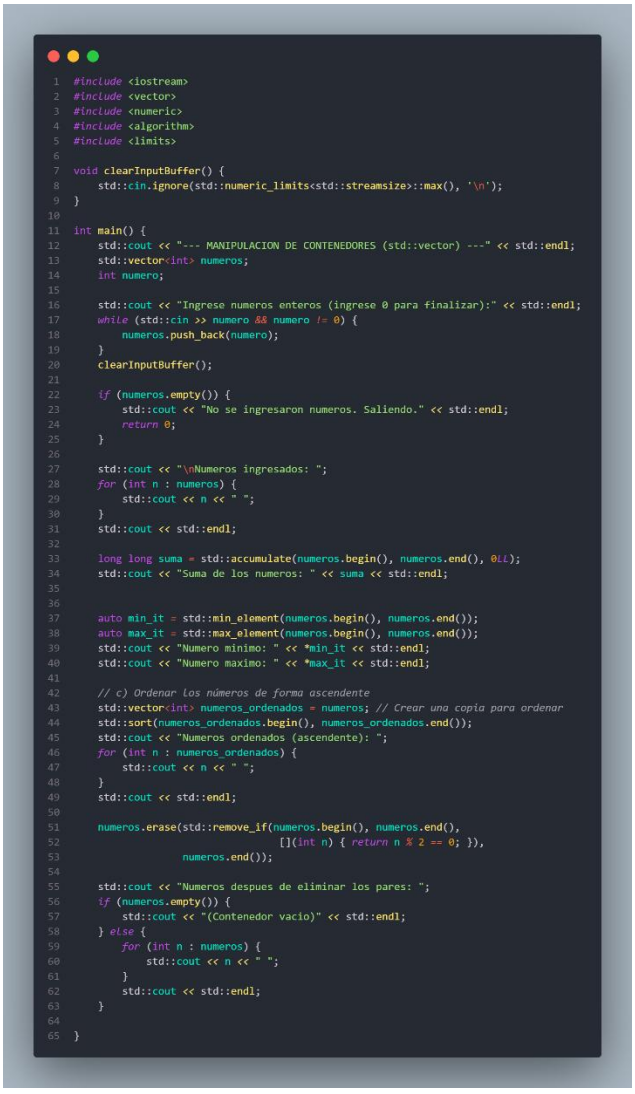
3. INFOGRAFÍA



Infografia 1

4. ACTIVIDADES

Experiencia de Práctica N° 01: Manipulación de Contenedores Objetivo: Utilizar los contenedores de la STL para almacenar y manipular datos. Descripción: Crear un programa que permita ingresar una lista de números enteros y almacenarlos en un vector. Luego, realizar operaciones como calcular la suma, encontrar el máximo y mínimo, ordenar de forma ascendente y eliminar los números pares. Mostrar los resultados y el contenido final del contenedor.

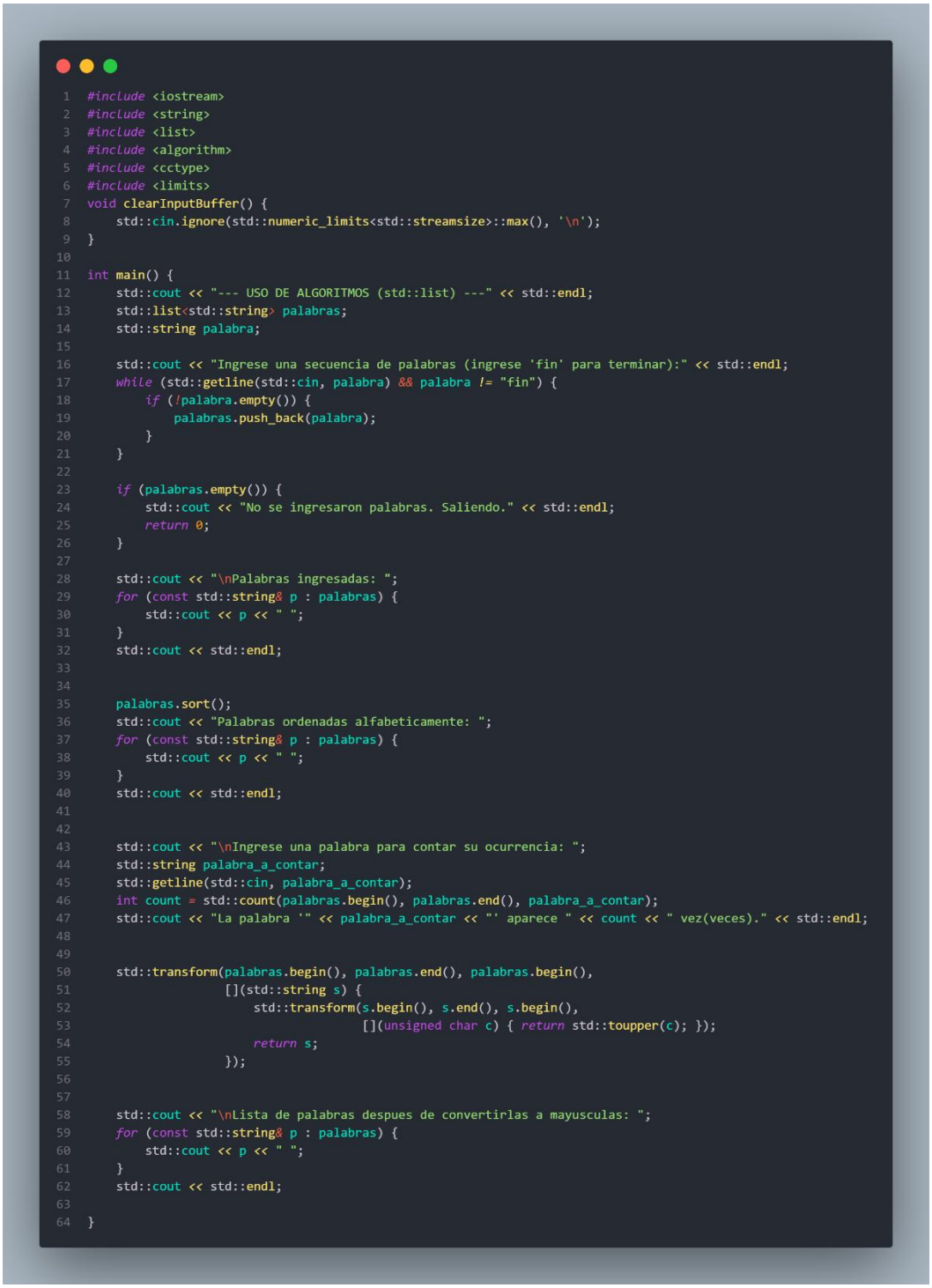


```cpp
#include <iostream>
#include <vector>
#include <numeric>
#include <algorithm>
#include <limits>

void clearInputBuffer() {
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

int main() {
    std::cout << "--- MANIPULACION DE CONTENEDORES (std::vector) ---" << std::endl;
    std::vector<int> numeros;
    int numero;

    std::cout << "Ingrese numeros enteros (ingrese 0 para finalizar):" << std::endl;
    while (std::cin >> numero && numero != 0) {
        numeros.push_back(numero);
    }
    clearInputBuffer();

    if (numeros.empty()) {
        std::cout << "No se ingresaron numeros. Saliendo." << std::endl;
        return 0;
    }

    std::cout << "\nNumeros ingresados: ";
    for (int n : numeros) {
        std::cout << n << " ";
    }
    std::cout << std::endl;

    long long suma = std::accumulate(numeros.begin(), numeros.end(), 0LL);
    std::cout << "Suma de los numeros: " << suma << std::endl;


    auto min_it = std::min_element(numeros.begin(), numeros.end());
    auto max_it = std::max_element(numeros.begin(), numeros.end());
    std::cout << "Numero minimo: " << *min_it << std::endl;
    std::cout << "Numero maximo: " << *max_it << std::endl;

    // c) Ordenar los números de forma ascendente
    std::vector<int> numeros_ordenados = numeros; // Crear una copia para ordenar
    std::sort(numeros_ordenados.begin(), numeros_ordenados.end());
    std::cout << "Numeros ordenados (ascendente): ";
    for (int n : numeros_ordenados) {
        std::cout << n << " ";
    }
    std::cout << std::endl;

    numeros.erase(std::remove_if(numeros.begin(), numeros.end(),
                                 [](int n) { return n % 2 == 0; }),
                  numeros.end());

    std::cout << "Numeros despues de eliminar los pares: ";
    if (numeros.empty()) {
        std::cout << "(Contenedor vacio)" << std::endl;
    } else {
        for (int n : numeros) {
            std::cout << n << " ";
        }
        std::cout << std::endl;
    }

}
```

Codigo N° 1

Experiencia de Práctica N° 02: Uso de Algoritmos Objetivo: Aplicar los algoritmos de la STL para resolver problemas específicos. Descripción: Implementar un programa que lea palabras ingresadas por el usuario y las almacene en una lista. Luego, utilizar los algoritmos `sort` para

ordenar, `count` para contar apariciones de una palabra, y `transform` para convertir las palabras a mayúsculas. Después, mostrar la lista modificada.



```cpp
#include <iostream>
#include <string>
#include <list>
#include <algorithm>
#include <cctype>
#include <limits>
void clearInputBuffer() {
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

int main() {
    std::cout << "--- USO DE ALGORITMOS (std::list) ---" << std::endl;
    std::list<std::string> palabras;
    std::string palabra;

    std::cout << "Ingrese una secuencia de palabras (ingrese 'fin' para terminar):" << std::endl;
    while (std::getline(std::cin, palabra) && palabra != "fin") {
        if (!palabra.empty()) {
            palabras.push_back(palabra);
        }
    }

    if (palabras.empty()) {
        std::cout << "No se ingresaron palabras. Saliendo." << std::endl;
        return 0;
    }

    std::cout << "\nPalabras ingresadas: ";
    for (const std::string& p : palabras) {
        std::cout << p << " ";
    }
    std::cout << std::endl;


    palabras.sort();
    std::cout << "Palabras ordenadas alfabeticamente: ";
    for (const std::string& p : palabras) {
        std::cout << p << " ";
    }
    std::cout << std::endl;


    std::cout << "\nIngrese una palabra para contar su ocurrencia: ";
    std::string palabra_a_contar;
    std::getline(std::cin, palabra_a_contar);
    int count = std::count(palabras.begin(), palabras.end(), palabra_a_contar);
    std::cout << "La palabra '" << palabra_a_contar << "' aparece " << count << " vez(veces)." << std::endl;


    std::transform(palabras.begin(), palabras.end(), palabras.begin(),
                   [](std::string s) {
                       std::transform(s.begin(), s.end(), s.begin(),
                                      [](unsigned char c) { return std::toupper(c); });
                       return s;
                   });


    std::cout << "\nLista de palabras despues de convertirlas a mayusculas: ";
    for (const std::string& p : palabras) {
        std::cout << p << " ";
    }
    std::cout << std::endl;

}
```

Codigo N° 2

Experiencia de Práctica N° 03: Manipulación de Iteradores Objetivo: Utilizar los iteradores para acceder y modificar elementos de los contenedores. Descripción: Crear un programa que permita ingresar una cadena de caracteres y almacenarla en un contenedor `deque`. Los estudiantes deben aprender a recorrer, modificar y eliminar elementos utilizando diferentes tipos de iteradores.

```cpp
#include <iostream>
#include <string>
#include <deque>
#include <algorithm>
#include <cctype>
#include <limits>


bool esVocal(char c) {
    c = std::tolower(c);
    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');
}

void clearInputBuffer() {
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

int main() {
    std::cout << "--- MANIPULACION DE ITERADORES (std::deque) ---" << std::endl;
    std::string cadena_str;
    std::cout << "Ingrese una cadena de caracteres: ";
    std::getline(std::cin, cadena_str);

    if (cadena_str.empty()) {
        std::cout << "Cadena vacia. Saliendo." << std::endl;
        return 0;
    }

    // Almacenar la cadena en un contenedor deque
    std::deque<char> caracteres(cadena_str.begin(), cadena_str.end());

    std::cout << "\nCadena original en deque: ";
    for (char c : caracteres) {
        std::cout << c;
    }
    std::cout << std::endl;

    // a) Utilizar un iterador para recorrer el contenedor y mostrar los caracteres uno por uno
    std::cout << "Recorriendo con iterador (caracter por caracter): ";
    for (std::deque<char>::iterator it = caracteres.begin(); it != caracteres.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    // b) Reemplazar todas las vocales en el contenedor con el carácter '*' utilizando un iterador
    std::cout << "Reemplazando vocales con '*':" << std::endl;
    for (std::deque<char>::iterator it = caracteres.begin(); it != caracteres.end(); ++it) {
        if (esVocal(*it)) {
            it = '';
        }
    }

    std::cout << "Cadena despues de reemplazar vocales: ";
    for (char c : caracteres) {
        std::cout << c;
    }
    std::cout << std::endl;

    std::cout << "Cadena en orden inverso (con iterador inverso): ";
    for (std::deque<char>::reverse_iterator rit = caracteres.rbegin(); rit != caracteres.rend(); ++rit) {
        std::cout << *rit;
    }
    std::cout << std::endl;

}
```

Codigo N° 3

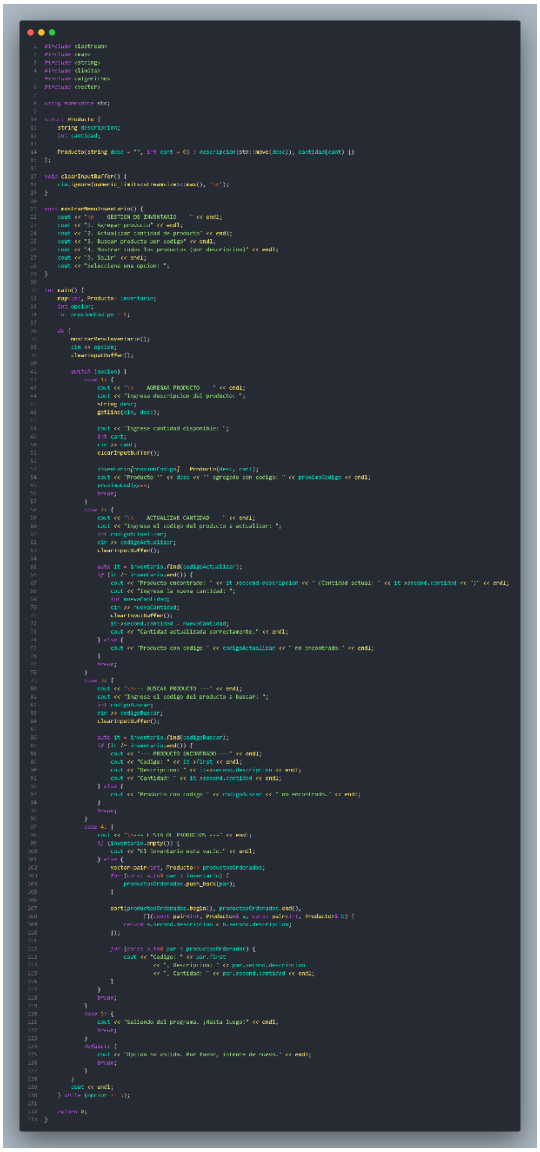
5. EJERCICIOS

Problema: Gestión de Inventario

Una empresa necesita gestionar su inventario de productos. Cada producto tiene un código único, una descripción y una cantidad disponible en stock.
El objetivo es desarrollar un programa que permita realizar las siguientes operaciones:

- a) Agregar un nuevo producto al inventario.
- b) Actualizar la cantidad disponible de un producto existente.
- c) Buscar un producto por su código.
- d) Mostrar la lista de productos ordenada alfabéticamente por su descripción.

Utiliza la STL para implementar la solución y proporciona una interfaz amigable para que los usuarios puedan interactuar con el programa.



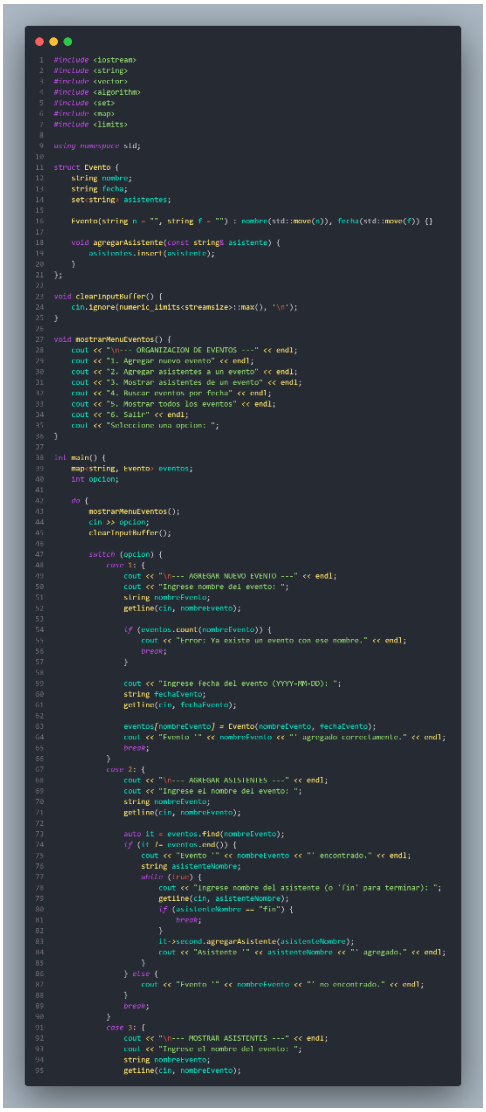
Codigo N° 4

2. Problema: Organización de Eventos

Un organizador de eventos necesita un sistema para gestionar la lista de asistentes a diferentes eventos. Cada evento tiene un nombre, una fecha y una lista de asistentes.
El objetivo es desarrollar un programa que permita realizar las siguientes operaciones:

- a) Agregar un nuevo evento a la lista.
- b) Agregar asistentes a un evento específico.
- c) Mostrar la lista de asistentes de un evento en orden alfabético.
- d) Buscar eventos por fecha.

Utiliza la STL para implementar la solución y proporciona una interfaz intuitiva para que el organizador pueda gestionar los eventos y los asistentes de manera eficiente.



```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <set>
#include <map>
#include <limits>

using namespace std;

struct Evento {
    string nombre;
    string fecha;
    set<string> asistentes;

    Evento(string n = "", string f = "") : nombre(std::move(n)), fecha(std::move(f)) {}

    void agregarAsistente(const string& asistente) {
        asistentes.insert(asistente);
    }
};

void clearInputBuffer() {
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}

void mostrarMenuEventos() {
    cout << "\n--- ORGANIZACION DE EVENTOS ---" << endl;
    cout << "1. Agregar nuevo evento" << endl;
    cout << "2. Agregar asistentes a un evento" << endl;
    cout << "3. Mostrar asistentes de un evento" << endl;
    cout << "4. Buscar eventos por fecha" << endl;
    cout << "5. Mostrar todos los eventos" << endl;
    cout << "6. Salir" << endl;
    cout << "Seleccione una opcion: ";
}

int main() {
    map<string, Evento> eventos;
    int opcion;

    do {
        mostrarMenuEventos();
        cin >> opcion;
        clearInputBuffer();

        switch (opcion) {
            case 1: {
                cout << "\n--- AGREGAR NUEVO EVENTO ---" << endl;
                cout << "Ingrese nombre del evento: ";
                string nombreEvento;
                getline(cin, nombreEvento);

                if (eventos.count(nombreEvento)) {
                    cout << "Error: Ya existe un evento con ese nombre." << endl;
                    break;
                }

                cout << "Ingrese fecha del evento (YYYY-MM-DD): ";
                string fechaEvento;
                getline(cin, fechaEvento);

                eventos[nombreEvento] = Evento(nombreEvento, fechaEvento);
                cout << "Evento '" << nombreEvento << "' agregado correctamente." << endl;
                break;
            }
            case 2: {
                cout << "\n--- AGREGAR ASISTENTES ---" << endl;
                cout << "Ingrese el nombre del evento: ";
                string nombreEvento;
                getline(cin, nombreEvento);

                auto it = eventos.find(nombreEvento);
                if (it != eventos.end()) {
                    cout << "Evento '" << nombreEvento << "' encontrado." << endl;
                    string asistenteNombre;
                    while (true) {
                        cout << "Ingrese nombre del asistente (o 'fin' para terminar): ";
                        getline(cin, asistenteNombre);
                        if (asistenteNombre == "fin") {
                            break;
                        }
                        it->second.agregarAsistente(asistenteNombre);
                        cout << "Asistente '" << asistenteNombre << "' agregado." << endl;
                    }
                } else {
                    cout << "Evento '" << nombreEvento << "' no encontrado." << endl;
                }
                break;
            }
            case 3: {
                cout << "\n--- MOSTRAR ASISTENTES ---" << endl;
                cout << "Ingrese el nombre del evento: ";
                string nombreEvento;
                getline(cin, nombreEvento);
```

Codigo N° 5

```
                auto it = eventos.find(nombreEvento);
                if (it != eventos.end()) {
                    cout << "Asistentes para el evento '" << it->second.nombre << "' (" << it->second.fecha << "):" << endl;
                    if (it->second.asistentes.empty()) {
                        cout << "No hay asistentes registrados para este evento." << endl;
                    } else {
                        for (const string& asistente : it->second.asistentes) {
                            cout << "- " << asistente << endl;
                        }
                    }
                } else {
                    cout << "Evento '" << nombreEvento << "' no encontrado." << endl;
                }
                break;
            }
            case 4: {
                cout << "\n--- BUSCAR EVENTOS POR FECHA ---" << endl;
                cout << "Ingrese la fecha a buscar (YYYY-MM-DD): ";
                string fechaBuscar;
                getline(cin, fechaBuscar);

                bool encontrados = false;
                cout << "Eventos en la fecha " << fechaBuscar << ":" << endl;
                for (const auto& par : eventos) {
                    if (par.second.fecha == fechaBuscar) {
                        cout << "- " << par.second.nombre << endl;
                        encontrados = true;
                    }
                }
                if (!encontrados) {
                    cout << "No se encontraron eventos para esa fecha." << endl;
                }
                break;
            }
            case 5: {
                cout << "\n--- LISTA DE TODOS LOS EVENTOS ---" << endl;
                if (eventos.empty()) {
                    cout << "No hay eventos registrados." << endl;
                } else {
                    for (const auto& par : eventos) {
                        cout << "Evento: " << par.second.nombre << ", Fecha: " << par.second.fecha << endl;
                        cout << "  Asistentes (" << par.second.asistentes.size() << "): ";
                        if (par.second.asistentes.empty()) {
                            cout << "Ninguno" << endl;
                        } else {
                            for (const auto& asistente : par.second.asistentes) {
                                cout << asistente << "; ";
                            }
                            cout << endl;
                        }
                    }
                }
                break;
            }
            case 6: {
                cout << "Saliendo del programa de organizacion de eventos. ¡Hasta luego!" << endl;
                break;
            }
            default: {
                cout << "Opcion no valida. Por favor, intente de nuevo." << endl;
                break;
            }
        }
    } while (opcion != 6);

    return 0;
}
```

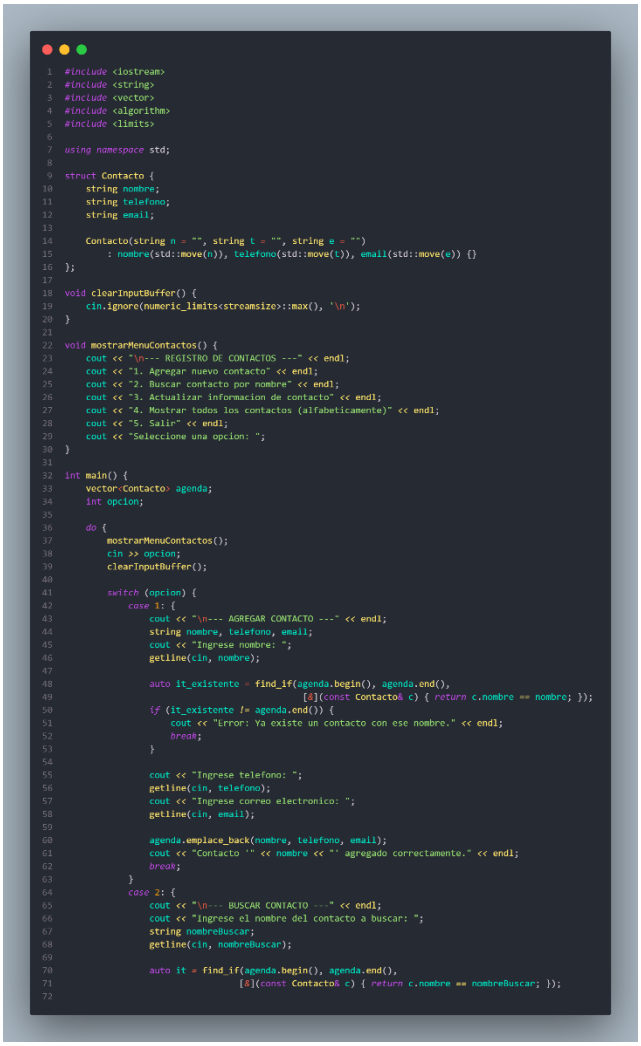
Codigo N° 6

6. 3. Problema: Registro de Contactos

Una agenda de contactos requiere un programa que permita almacenar y gestionar los datos de contactos de una persona. Cada contacto tiene un nombre, número de teléfono y dirección de correo electrónico.
El objetivo es desarrollar un programa que permita realizar las siguientes operaciones:

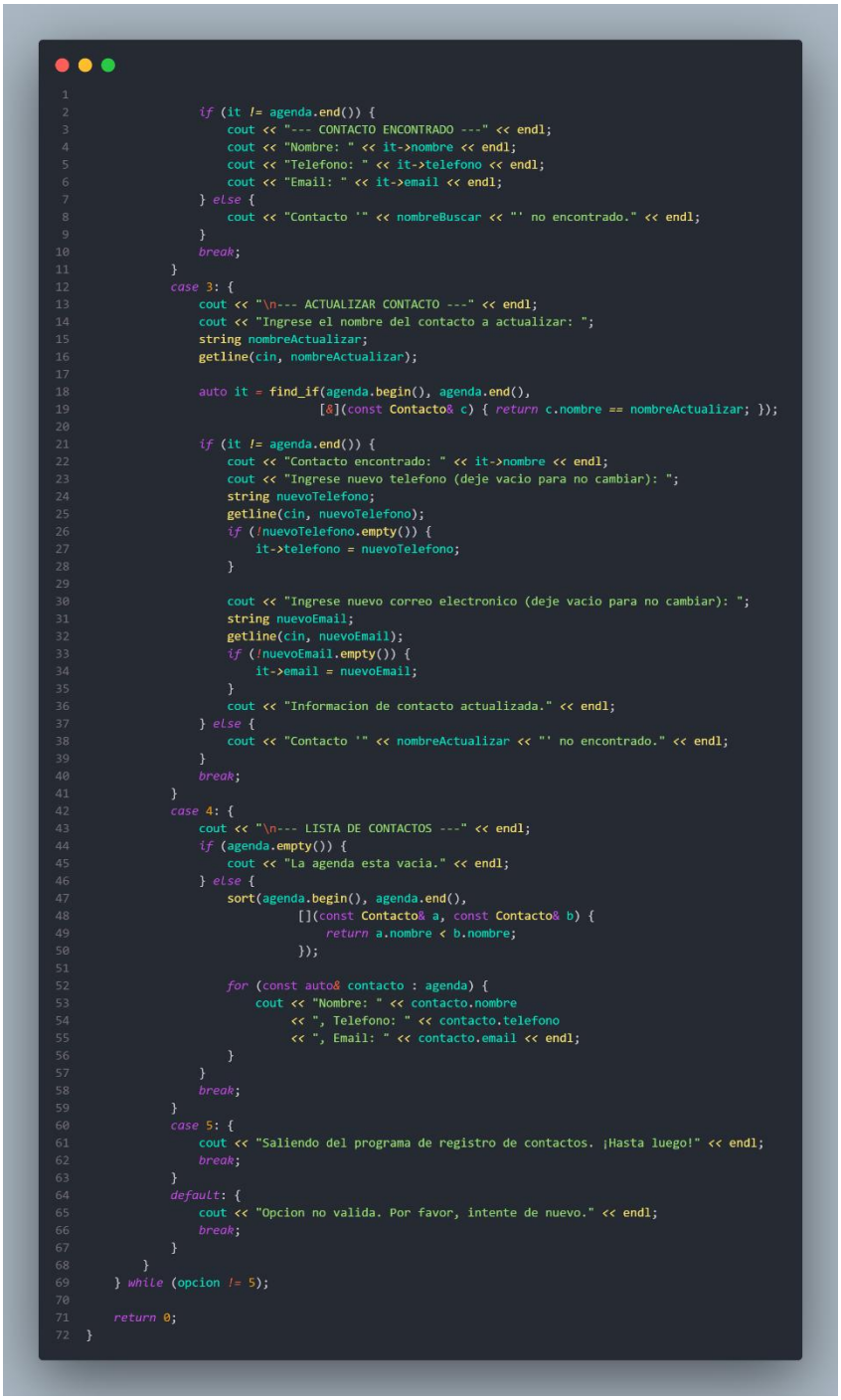
- a) Agregar un nuevo contacto a la agenda.
- b) Buscar un contacto por su nombre.
- c) Actualizar la información de un contacto existente.
- d) Mostrar la lista de contactos ordenada alfabéticamente.

Utiliza la STL para implementar la solución y proporciona una interfaz sencilla para que los usuarios puedan administrar eficazmente su lista de contactos.



```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <limits>

using namespace std;

struct Contacto {
    string nombre;
    string telefono;
    string email;

    Contacto(string n = "", string t = "", string e = "")
        : nombre(std::move(n)), telefono(std::move(t)), email(std::move(e)) {}
};

void clearInputBuffer() {
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}

void mostrarMenuContactos() {
    cout << "\n--- REGISTRO DE CONTACTOS ---" << endl;
    cout << "1. Agregar nuevo contacto" << endl;
    cout << "2. Buscar contacto por nombre" << endl;
    cout << "3. Actualizar informacion de contacto" << endl;
    cout << "4. Mostrar todos los contactos (alfabeticamente)" << endl;
    cout << "5. Salir" << endl;
    cout << "Seleccione una opcion: ";
}

int main() {
    vector<Contacto> agenda;
    int opcion;

    do {
        mostrarMenuContactos();
        cin >> opcion;
        clearInputBuffer();

        switch (opcion) {
            case 1: {
                cout << "\n--- AGREGAR CONTACTO ---" << endl;
                string nombre, telefono, email;
                cout << "Ingrese nombre: ";
                getline(cin, nombre);

                auto it_existente = find_if(agenda.begin(), agenda.end(),
                                            [&](const Contacto& c) { return c.nombre == nombre; });
                if (it_existente != agenda.end()) {
                    cout << "Error: Ya existe un contacto con ese nombre." << endl;
                    break;
                }

                cout << "Ingrese telefono: ";
                getline(cin, telefono);
                cout << "Ingrese correo electronico: ";
                getline(cin, email);

                agenda.emplace_back(nombre, telefono, email);
                cout << "Contacto '" << nombre << "' agregado correctamente." << endl;
                break;
            }
            case 2: {
                cout << "\n--- BUSCAR CONTACTO ---" << endl;
                cout << "Ingrese el nombre del contacto a buscar: ";
                string nombreBuscar;
                getline(cin, nombreBuscar);

                auto it = find_if(agenda.begin(), agenda.end(),
                                  [&](const Contacto& c) { return c.nombre == nombreBuscar; });

```

Codigo N° 7



```
                if (it != agenda.end()) {
                    cout << "--- CONTACTO ENCONTRADO ---" << endl;
                    cout << "Nombre: " << it->nombre << endl;
                    cout << "Telefono: " << it->telefono << endl;
                    cout << "Email: " << it->email << endl;
                } else {
                    cout << "Contacto '" << nombreBuscar << "' no encontrado." << endl;
                }
                break;
            }
            case 3: {
                cout << "\n--- ACTUALIZAR CONTACTO ---" << endl;
                cout << "Ingrese el nombre del contacto a actualizar: ";
                string nombreActualizar;
                getline(cin, nombreActualizar);

                auto it = find_if(agenda.begin(), agenda.end(),
                                  [&](const Contacto& c) { return c.nombre == nombreActualizar; });

                if (it != agenda.end()) {
                    cout << "Contacto encontrado: " << it->nombre << endl;
                    cout << "Ingrese nuevo telefono (deje vacio para no cambiar): ";
                    string nuevoTelefono;
                    getline(cin, nuevoTelefono);
                    if (!nuevoTelefono.empty()) {
                        it->telefono = nuevoTelefono;
                    }

                    cout << "Ingrese nuevo correo electronico (deje vacio para no cambiar): ";
                    string nuevoEmail;
                    getline(cin, nuevoEmail);
                    if (!nuevoEmail.empty()) {
                        it->email = nuevoEmail;
                    }
                    cout << "Informacion de contacto actualizada." << endl;
                } else {
                    cout << "Contacto '" << nombreActualizar << "' no encontrado." << endl;
                }
                break;
            }
            case 4: {
                cout << "\n--- LISTA DE CONTACTOS ---" << endl;
                if (agenda.empty()) {
                    cout << "La agenda esta vacia." << endl;
                } else {
                    sort(agenda.begin(), agenda.end(),
                              [](const Contacto& a, const Contacto& b) {
                                  return a.nombre < b.nombre;
                              });

                    for (const auto& contacto : agenda) {
                        cout << "Nombre: " << contacto.nombre
                             << ", Telefono: " << contacto.telefono
                             << ", Email: " << contacto.email << endl;
                    }
                }
                break;
            }
            case 5: {
                cout << "Saliendo del programa de registro de contactos. ¡Hasta luego!" << endl;
                break;
            }
            default: {
                cout << "Opcion no valida. Por favor, intente de nuevo." << endl;
                break;
            }
        }
    } while (opcion != 5);

    return 0;
}
```

Codigo N° 8

7. CUESTIONARIO

1. ¿Qué significa STL en C++?
STL significa *Standard Template Library*, una biblioteca de plantillas que proporciona estructuras de datos y algoritmos genéricos.

2. ¿Cuáles son los componentes principales de la STL?
Los principales componentes son: contenedores, algoritmos e iteradores.

3. ¿Cuál es la utilidad de los contenedores en la STL?
Permiten almacenar y gestionar colecciones de datos de manera eficiente y flexible.

4. ¿Qué tipo de contenedor de la STL se utiliza cuando se necesita una colección ordenada de elementos únicos?
El contenedor `set`.

5. Menciona tres ejemplos de contenedores asociativos de la STL.
`set`, `map` y `unordered_map`.

6. ¿Cuál es la diferencia entre los contenedores vector y list en la STL?
`vector` almacena elementos en un arreglo dinámico contiguo y permite acceso rápido por índice, mientras que `list` es una lista doblemente enlazada, más eficiente para inserciones/borrados en cualquier posición.

7. ¿Qué es un iterador en la STL?
Es un objeto que permite recorrer los elementos de un contenedor de forma similar a un puntero.

8. ¿Cuál es la función de los algoritmos en la STL?
Realizar operaciones sobre rangos de datos como búsqueda, ordenamiento, conteo, copia, etc.

9. Menciona tres ejemplos de algoritmos disponibles en la STL.
`sort`, `find`, `count`.

10. ¿Cuál es la ventaja de utilizar los algoritmos de la STL en lugar de implementarlos manualmente?
Mayor eficiencia, menor posibilidad de errores y ahorro de tiempo en la programación.

11. ¿Qué es un iterador de inserción en la STL y cómo se utiliza?
Es un iterador especial que permite insertar elementos en un contenedor durante operaciones como `copy`. Ej.: `back_inserter(vec)`.

12. ¿Cuál es la diferencia entre un iterador constante y un iterador normal en la STL?
El iterador constante (`const_iterator`) no permite modificar el valor apuntado; el normal (`iterator`) sí.

13. ¿Qué contenedor de la STL se utiliza para almacenar elementos en pares clave-valor?
`map` o `unordered_map`.

14. Menciona tres ejemplos de algoritmos numéricos disponibles en la STL.
`accumulate`, `adjacent_difference`, `partial_sum`.

15. ¿Cuál es la utilidad de los iteradores de flujo en la STL?
Permiten leer y escribir datos directamente en flujos de entrada/salida como archivos o consola.

16. ¿Cuál es la función del adaptador de iterador reverse_iterator en la STL?
Permite recorrer un contenedor en sentido inverso.

17. ¿Qué contenedor de la STL se utiliza para almacenar elementos en orden de inserción, búsquedas y duplicados de forma rápida?
`multiset` o `unordered_multiset`.

18. ¿Cuál es la diferencia entre el contenedor set y el contenedor map en la STL?
`set` almacena solo claves únicas, mientras que `map` almacena pares clave-valor con claves únicas.

19. ¿Qué es el contenedor queue en la STL y cuándo se utiliza?
Es una estructura FIFO (primero en entrar, primero en salir), útil para gestionar tareas en orden.

20. ¿Cuál es la función del contenedor queue en la STL y cómo se implementa?
Sirve para manejar datos en orden FIFO y se implementa con `std::queue<T>`.

21. ¿Cuál es la función del contenedor list en la STL y cómo se utiliza?
Permite inserciones/borrados eficientes en cualquier parte de la lista. Se usa como `std::list<T>`.

22. ¿Qué es la función lambda en la STL y cuál es su utilidad?
Es una función anónima usada comúnmente para pasar funciones a algoritmos de STL.

23. ¿Cuál es la diferencia entre vector y stack en la STL y cómo se implementa?
`vector` es un arreglo dinámico, `stack` es LIFO (último en entrar, primero en salir). `stack` se implementa como `std::stack<T>`.

24. ¿Qué es un iterador con valor y cómo se utiliza en la STL?
No es un término estándar en STL. Probablemente se refiere a un iterador que permite acceso y modificación del valor.

25. ¿Cuál es el algoritmo sort de la STL y cómo se utiliza?
`sort` ordena un rango de elementos. Ej.: `std::sort(vec.begin(), vec.end());`