Informe de Laboratorio 05

Tema: C++ Templates, Qt / Widgets parte II

		Nota

Estudiante	Escuela	Asignatura
Escobedo Ocaña Jorge Luis	Escuela Profesional de Ingeniería de Sistemas	Tecnología de Objetos Semestre: VI
Laboratorio	**Tema**	**Duración**
05	C++ Templates, Qt / Widgets parte II	02 horas
Semestre académico	**Fecha de inicio**	**Fecha de entrega**
2024 - B	24/10/2024	31/10/2024

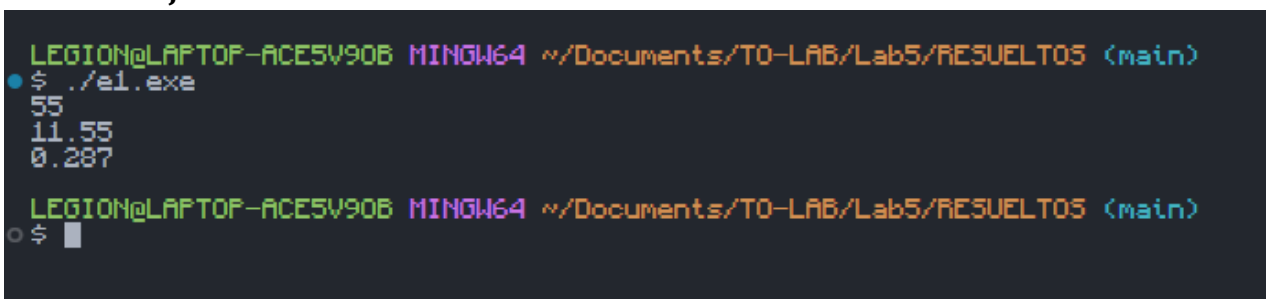
1. **Tarea**
   - **a.** Desarrollar los ejercicios presentados en la guía de laboratorio 04.
   - **b.** Utilizar Git para evidenciar su trabajo.
   - **c.** Enviar trabajo al profesor en un repositorio GitHub Privado, dándole permisos como colaborador.
2. **Equipos, materiales y temas utilizados**
   - **a.** Sistema Operativo: Windows.
   - **b.** Compilador: MinGW (GCC-6.3.0-1) o MSVC.
   - **c.** Entorno de Desarrollo Integrado (IDE): Qt Creator.
   - **d.** Versiones de C++: C++17 y C++20.
   - **e.** Bibliotecas: Bibliotecas estándar de C++ (STL), Qt Widgets, Qt Core, Qt Network, Qt Multimedia.
   - **f.** Documentación y Recursos: Documentación de Qt, Tutoriales en línea.
   - **g.** Repositorio en GITHUB: https://github.com/Juerges28/TO-LAB/tree/main/Lab5/
3. **Ejercicios Resueltos**
   - **3.1. PLANTILLAS**
   - **a. Actividad 1**

     Crear una función plantilla que permita realizar la aritmética de 2 números sin importar el tipo input (entero, flotante, doble, o similar)

```
C/C++
#include <iostream>

using namespace std;

template <typename mitipo>
```

```
mitipo aritmetica(mitipo x, mitipo y){
    return x + y;
}

int main(){
    cout << aritmetica(10, 45) << endl;
    cout << aritmetica(3.45, 8.10) << endl;
    cout << aritmetica(0.14f, 0.147f) << endl;
    return 0;
}
```

Ejecución:

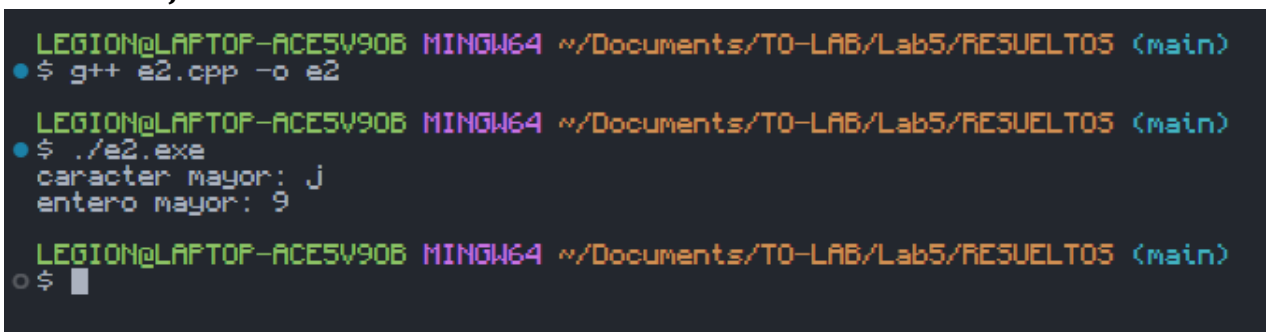


b. Actividad 2

Crear una función plantilla que devuelva el carácter 'mayor' (según orden alfabético), o el número mayor sin importar el tipo input.

```
C/C++
#include <iostream>

using namespace std;

template <typename Tipo> Tipo mayorDe(Tipo a, Tipo b) {
    return (a > b ? a : b);
}

int main(){
    char x = mayorDe<char>('j', 'a');
    int y = mayorDe<int>(9, 4);

    cout << "caracter mayor: " << x << endl;
    cout << "entero mayor: " << y << endl;
}
```

Ejecución:



c. Actividad 3

Clase plantilla para insertar cualquier tipo de dato numérico en una posición específica de un array..

```C/C++
#include <iostream>

using namespace std;

template<typename T>
class Contenedor {
    public:
        T metodo1(int xx){
            return array[xx];
        }
        void metodo2(T val, int i){
            array[i] = val;
        }
        void mostrar() const;
    private:
        T array[4];
};

template<typename T>
void Contenedor<T>::mostrar() const {
    for (int i = 0; i < 4; i++) {
        cout << array[i] << " ";
    }
}

int main() {
    Contenedor<int> z;
    z.metodo2(10, 0);
    z.metodo2(50, 1);
    z.metodo2(60, 2);
    z.metodo2(70, 3);
```

```
    z.mostrar();
    return 0;
}
```

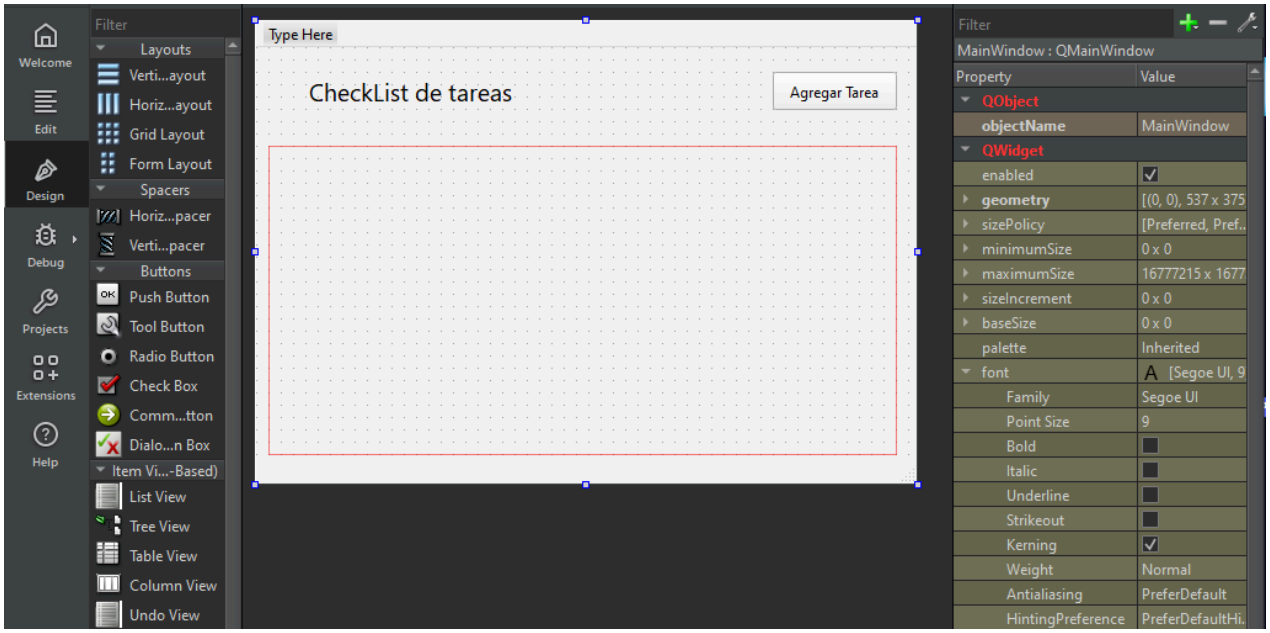
Ejecución

```
LEGION@LAPTOP-ACE5V90B MINGW64 ~/Documents/TO-LAB/Lab5/RESUELTOS (main)
$ g++ e3.cpp -o e3

LEGION@LAPTOP-ACE5V90B MINGW64 ~/Documents/TO-LAB/Lab5/RESUELTOS (main)
$ ./e3.exe
10 50 60 70
LEGION@LAPTOP-ACE5V90B MINGW64 ~/Documents/TO-LAB/Lab5/RESUELTOS (main)
$
```

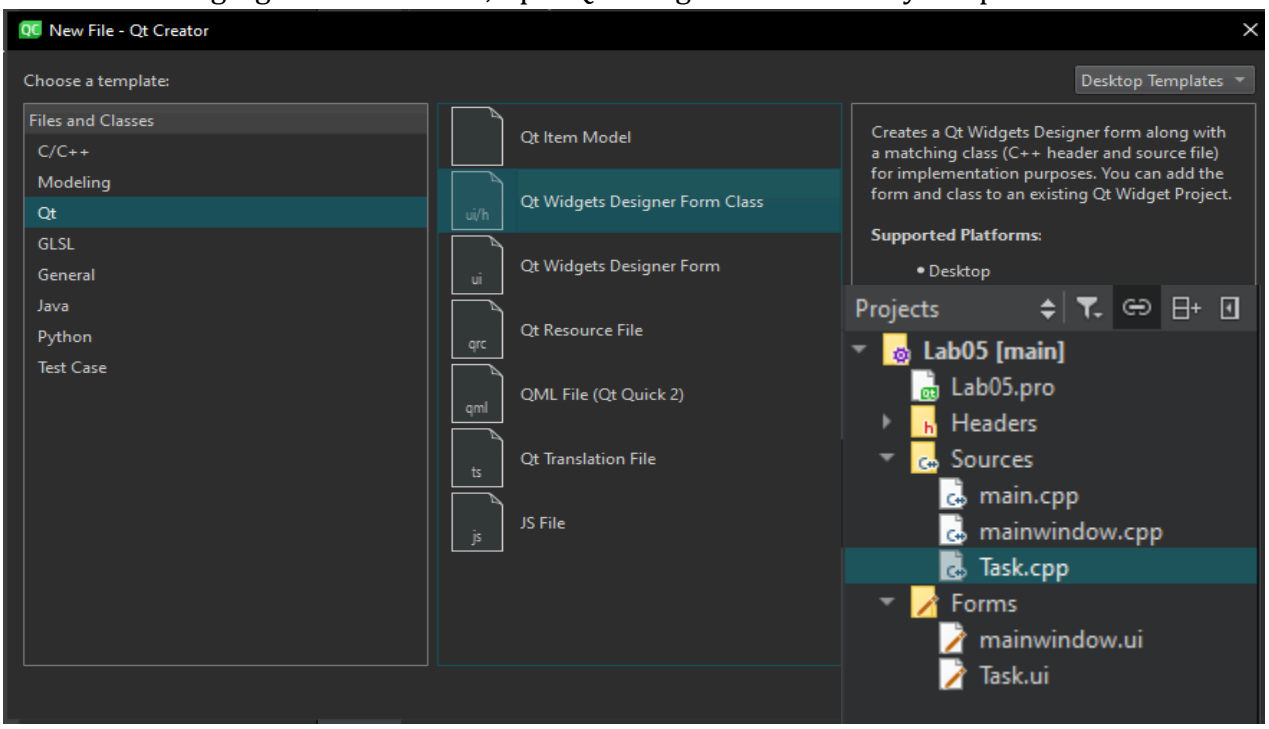
3.2. APLICACIÓN QT WIDGETS, SIGNALS & SLOTS EN OBJETOS DE FORMULARIO GUÍA

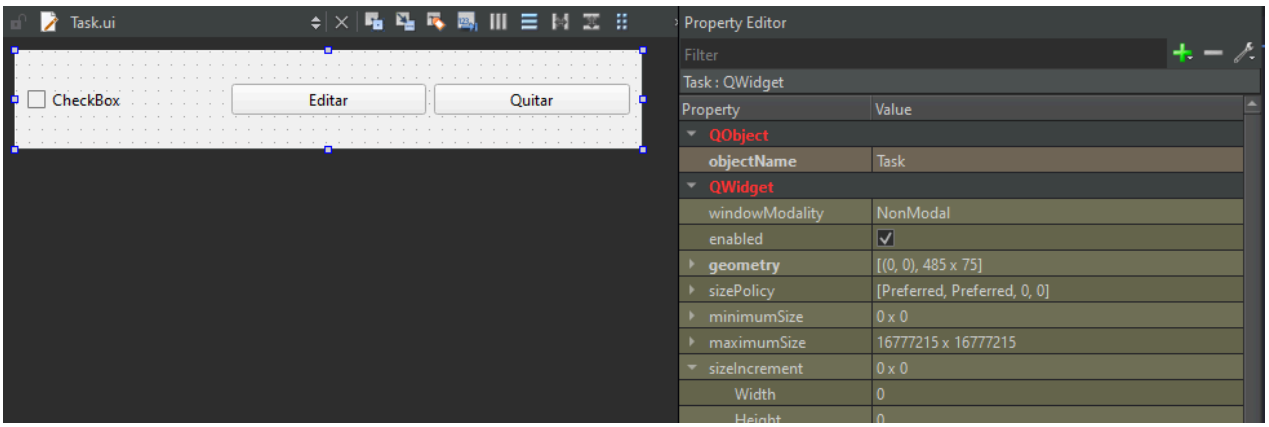
a. Actividad 1

Editar el archivo MainWindow.ui
Agregar los siguientes objetos:
Label → statusLabel propiedad text: Checklist de tareas
Button → addTaskButton propiedad text: Agregar tarea
Vertical layout → tasksLayout propiedad text:



Agregar la clase 'Task', tipo "Qt Dedigner form class" y template WIDGET.





mainwindow.h

```C/C++
#pragma once

#include <QMainWindow>
#include <QVector>
#include "Task.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow {
```

```
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

public slots:
    void addTask();
    void removeTask(Task* task);

private:
    Ui::MainWindow *ui;
    QVector<Task*> tasks;
};
```

Task.h

```
C/C++
#pragma once

#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui { class Task; }
QT_END_NAMESPACE

class Task : public QWidget {
    Q_OBJECT

public:
    explicit Task(QWidget *parent = nullptr);
    ~Task();

signals:
    void removed(Task* task);

private:
    Ui::Task *ui;
};
```

Task.cpp

```
C/C++
#include "Task.h"
#include "ui_Task.h"

Task::Task(QWidget *parent) :
```

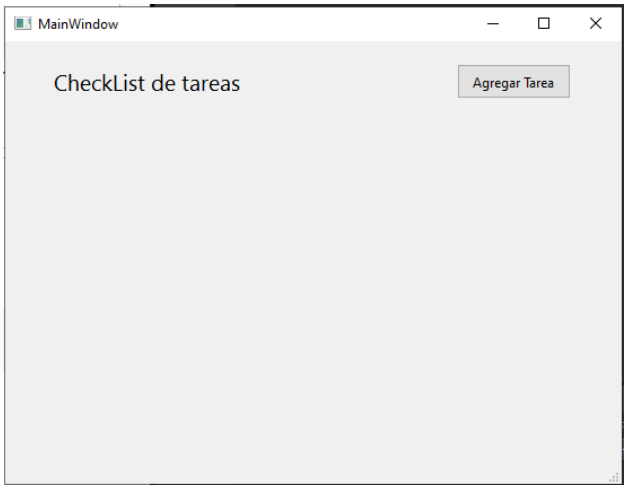
```
    QWidget(parent), ui(new Ui::Task) {
    ui->setupUi(this);

    connect(ui->removeButton, &QPushButton::clicked, this, [this]() {
        emit removed(this);
    });

    setMinimumHeight(50);
    setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
}

Task::~Task() {
    delete ui;
}
```

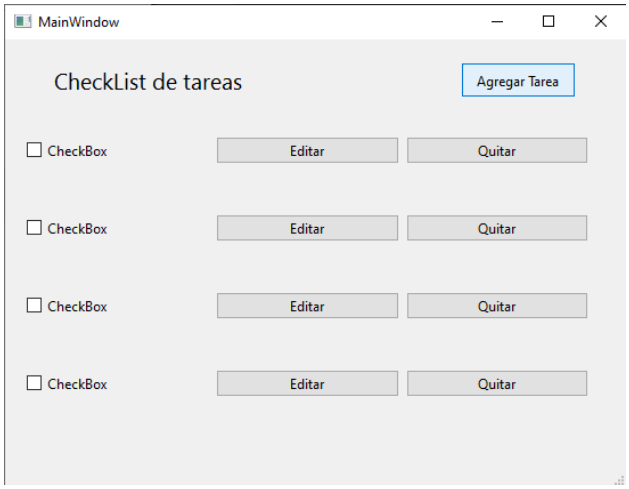
mainwindow.cpp

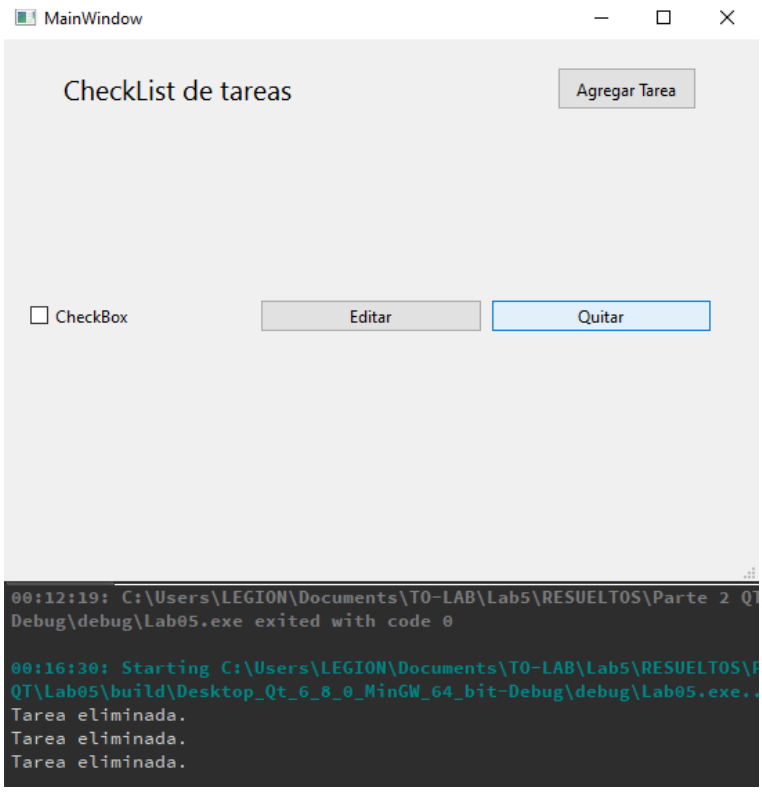
```
C/C++
#include "MainWindow.h"
#include "ui_MainWindow.h"
#include "Task.h"
#include <QVBoxLayout>
#include <QDebug>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow), tasks() {
    ui->setupUi(this);

    connect(ui->addTaskButton, &QPushButton::clicked, this,
&MainWindow::addTask);
}

MainWindow::~MainWindow() {
    delete ui;
}

void MainWindow::addTask() {
    Task *task = new Task(this);
    ui->tasksLayout->addWidget(task);
    tasks.append(task);
    connect(task, &Task::removed, this, &MainWindow::removeTask);
    ui->tasksLayout->update();
}

void MainWindow::removeTask(Task* task) {
    ui->tasksLayout->removeWidget(task);
    tasks.removeOne(task);
    task->deleteLater();
    qDebug() << "Tarea eliminada.";
```

```
}
```

Ejecución:


MainWindow
CheckList de tareas
Agregar Tarea


MainWindow
CheckList de tareas
Agregar Tarea
CheckBox
Editar
Quitar
CheckBox
Editar
Quitar
CheckBox
Editar
Quitar
CheckBox
Editar
Quitar


MainWindow
CheckList de tareas
Agregar Tarea
CheckBox
Editar
Quitar
00:12:19: C:\Users\LEGION\Documents\TO-LAB\Lab5\RESUELTOS\Parte 2 QT
Debug\debug\Lab05.exe exited with code 0
00:16:30: Starting C:\Users\LEGION\Documents\TO-LAB\Lab5\RESUELTOS\P
QT\Lab05\build\Desktop_Qt_6_8_0_MinGW_64_bit-Debug\debug\Lab05.exe..
Tarea eliminada.
Tarea eliminada.
Tarea eliminada.

4. Ejercicios Propuestos

a. Actividad 1

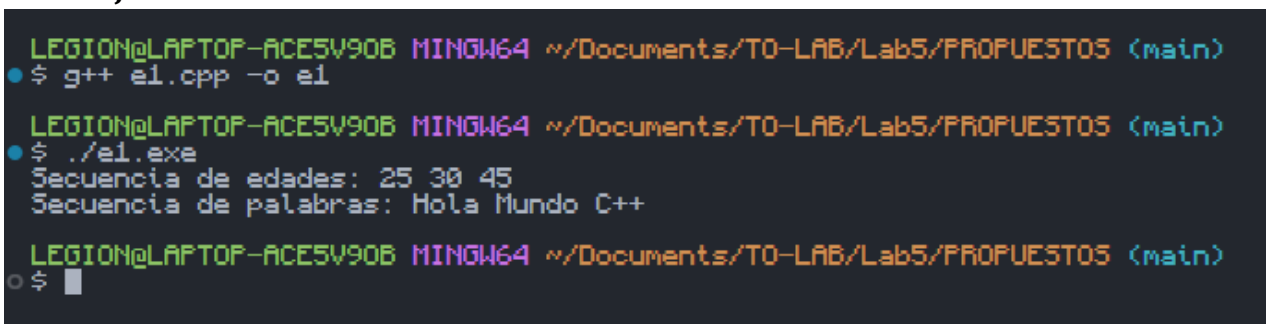
Usar clase template para implementar estructuras de lista enlazada simple, que permita armar secuencia de edades, secuencia de palabras.

C/C++

```
#include <iostream>
#include <string>
using namespace std;

template <typename T>
class Nodo {
public:
    T dato;
    Nodo<T>* siguiente;

    Nodo(T valor) : dato(valor), siguiente(nullptr) {}
};

template <typename T>
class ListaEnlazadaSimple {
private:
    Nodo<T>* cabeza;

public:
    ListaEnlazadaSimple() : cabeza(nullptr) {}

    void agregar(T valor) {
        Nodo<T>* nuevoNodo = new Nodo<T>(valor);
        if (!cabeza) {
            cabeza = nuevoNodo;
        } else {
            Nodo<T>* temp = cabeza;
            while (temp->siguiente) {
                temp = temp->siguiente;
            }
            temp->siguiente = nuevoNodo;
        }
    }

    void imprimir() const {
        Nodo<T>* temp = cabeza;
        while (temp) {
            cout << temp->dato << " ";
            temp = temp->siguiente;
        }
        cout << endl;
    }

    ~ListaEnlazadaSimple() {
```

```
        Nodo<T>* temp;
        while (cabeza) {
            temp = cabeza;
            cabeza = cabeza->siguiente;
            delete temp;
        }
    }
};

int main() {
    ListaEnlazadaSimple<int> listaEdades;
    listaEdades.agregar(25);
    listaEdades.agregar(30);
    listaEdades.agregar(45);
    cout << "Secuencia de edades: ";
    listaEdades.imprimir();

    ListaEnlazadaSimple<string> listaPalabras;
    listaPalabras.agregar("Hola");
    listaPalabras.agregar("Mundo");
    listaPalabras.agregar("C++");
    cout << "Secuencia de palabras: ";
    listaPalabras.imprimir();

    return 0;
}
```

Ejecución:



b. Actividad 2

Transcribir y analizar el siguiente código, describir el comportamiento.

Transcripción del código

```
C/C++
#include <iostream>
using namespace std;
template <class T>
```

```cpp
class Contendor
{
    T elemento;

public:
    Contendor(T arg) { elemento = arg; }
    T add() { return ++elemento; }
};
template <>
class Contendor<char>
{
    char elemento;

public:
    Contendor(char arg) { elemento = arg; }
    char uppercase()
    {
        if ((elemento >= 'a') && (elemento <= 'z'))
            elemento += 'A' - 'a';
        return elemento;
    }
};
int main()
{
    Contendor<int> cint(5);
    Contendor<char> cchar('c');
    std::cout << cint.add() << std::endl;
    std::cout << cchar.uppercase() << std::endl;
    return 0;
}
```

Ejecución:

```
LEGION@LAPTOP-ACE5V90B MINGW64 ~/Documents/TO-LAB/Lab5/PROPUESTOS (main)
$ g++ e2.cpp -o e2

LEGION@LAPTOP-ACE5V90B MINGW64 ~/Documents/TO-LAB/Lab5/PROPUESTOS (main)
$ ./e2.exe
6
C

LEGION@LAPTOP-ACE5V90B MINGW64 ~/Documents/TO-LAB/Lab5/PROPUESTOS (main)
$
```

Análisis:

El código define una clase template llamada Contenedor, esta clase permite almacenar un elemento de tipo T (generico). Almacena un valor tipo T que se inicializa a través del constructor, también la clase tiene un método add() el cual incrementa y devuelve el valor del elemento usando "++".

El código también incluye un caso especial con el tipo char. Se modifica el comportamiento de la clase, en este caso en lugar del método add() tenemos el método uppercase() que convierte al carácter almacenado en mayúscula si es una minúscula. Para hacer esto uppercase() comprueba si el carácter está entre 'a' y 'z' si es así convierte elemento a mayúscula sumando la diferencia entre los valores ASCII de 'A' y 'a'. Finalmente, devuelve el carácter en su forma convertida.

En la función main() se crean instancias de Contenedor para ambos casos int y char, se llama a los métodos correspondientes de cada caso, add() y uppercase() y al ejecutarse se comprueba la funcionalidad antes descrita.

c. Actividad 3

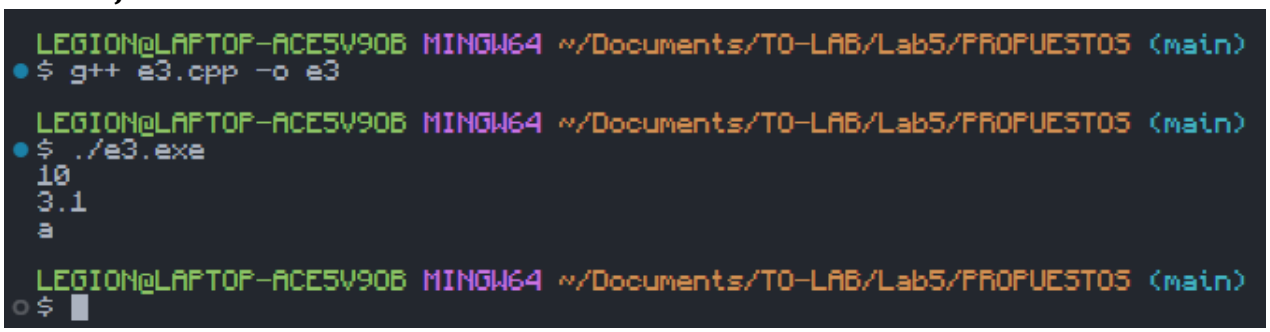
Transcribir y analizar el siguiente código, describir el comportamiento.

Transcripción del código

C/C++

```
#include <iostream>
using namespace std;
template <class T = char, int N = 8>
class Comun1
{
    T bloque[N];

public:
    void set(int num, T tval);
    T get(int num);
};
template <class T, int N>
void Comun1<T, N>::set(int num, T tval)
{
    bloque[num] = tval;
}
template <class T, int N>
T Comun1<T, N>::get(int num)
{
    return bloque[num];
}
int main()
{
    Comun1<int, 5> objInt;
    Comun1<double, 5> objFloat;
    Comun1<> obj;
    objInt.set(0, 10);
    objFloat.set(2, 3.1);
    obj.set(4, 'a');
    cout << objInt.get(0) << std::endl;
    cout << objFloat.get(2) << std::endl;
    cout << obj.get(4) << std::endl;
    return 0;
}
```

Ejecución:



Análisis:

El código define una clase template llamada Comun1, que permite almacenar elementos en un array de tipo genérico T y tamaño N, con valores predeterminados de char y 8, respectivamente. La clase incluye métodos para establecer set() y obtener get() valores en posiciones específicas del array.

En la función main(), se crean instancias de Comun1 para diferentes tipos de datos: objInt para enteros, objFloat para flotantes y obj utilizando los valores predeterminados. Se almacenan y recuperan varios valores, imprimiendo 10, 3.1 y 'a' en la salida. Esto demuestra la flexibilidad de la clase para manejar distintos tipos de datos y tamaños de arrays mediante el uso de templates.

d. Actividad 4

Usar una clase plantilla para estructuras pila, debe permitir manipular inserciones de tipos numéricos o caracteres.

```C/C++
#include <iostream>
using namespace std;

template <typename T>
class Pila {
private:
    T* elementos;
    int capacidad;
    int cima;

public:
    Pila(int size);
    ~Pila();
    void push(T elemento);
    T pop();
    T peek();
    bool isEmpty();
    bool isFull();
};
```

```cpp
template <typename T>
Pila<T>::Pila(int size) {
    capacidad = size;
    elementos = new T[capacidad];
    cima = -1;
}

template <typename T>
Pila<T>::~Pila() {
    delete[] elementos;
}

template <typename T>
void Pila<T>::push(T elemento) {
    if (!isFull()) {
        elementos[++cima] = elemento;
    } else {
        cout << "La pila está llena." << endl;
    }
}

template <typename T>
T Pila<T>::pop() {
    if (!isEmpty()) {
        return elementos[cima--];
    } else {
        cout << "La pila está vacía." << endl;
        return T();
    }
}

template <typename T>
T Pila<T>::peek() {
    if (!isEmpty()) {
        return elementos[cima];
    } else {
        cout << "La pila está vacía." << endl;
        return T();
    }
}

template <typename T>
bool Pila<T>::isEmpty() {
    return cima == -1;
}


template <typename T>
bool Pila<T>::isFull() {
    return cima == capacidad - 1;
}
```

```
int main() {
    Pila<int> pilaInt(5);
    pilaInt.push(1);
    pilaInt.push(2);
    cout << "Elemento en la cima (int): " << pilaInt.peek() << endl;
    cout << "Elemento removido (int): " << pilaInt.pop() << endl;
    cout << "Elemento en la cima (int) después de pop: " << pilaInt.peek() <<
endl;

    Pila<char> pilaChar(5);
    pilaChar.push('A');
    pilaChar.push('B');
    cout << "Elemento en la cima (char): " << pilaChar.peek() << endl;
    cout << "Elemento removido (char): " << pilaChar.pop() << endl;
    cout << "Elemento en la cima (char) después de pop: " << pilaChar.peek() <<
endl;

    return 0;
}
```

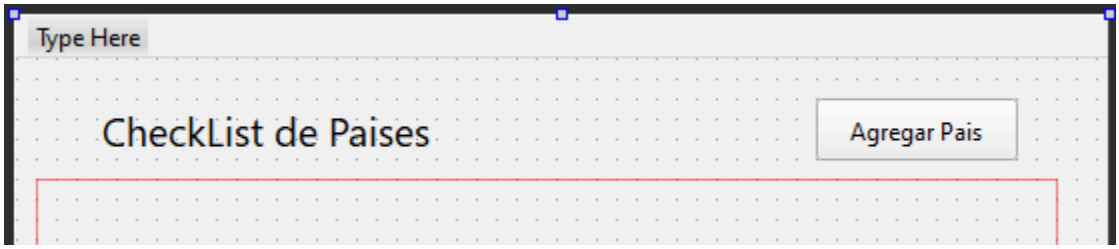
Ejecución:

```
LEGION@LAPTOP-ACE5V90B MINGW64 ~/Documents/TO-LAB/Lab5/PROPUESTOS (main)
$ g++ e4.cpp -o e4

LEGION@LAPTOP-ACE5V90B MINGW64 ~/Documents/TO-LAB/Lab5/PROPUESTOS (main)
$ ./e4.exe
Elemento en la cima (int): 2
Elemento removido (int): 2
Elemento en la cima (int) despu├®s de pop: 1
Elemento en la cima (char): B
Elemento removido (char): B
Elemento en la cima (char) despu├®s de pop: A

LEGION@LAPTOP-ACE5V90B MINGW64 ~/Documents/TO-LAB/Lab5/PROPUESTOS (main)
$
```

e. Actividad 5

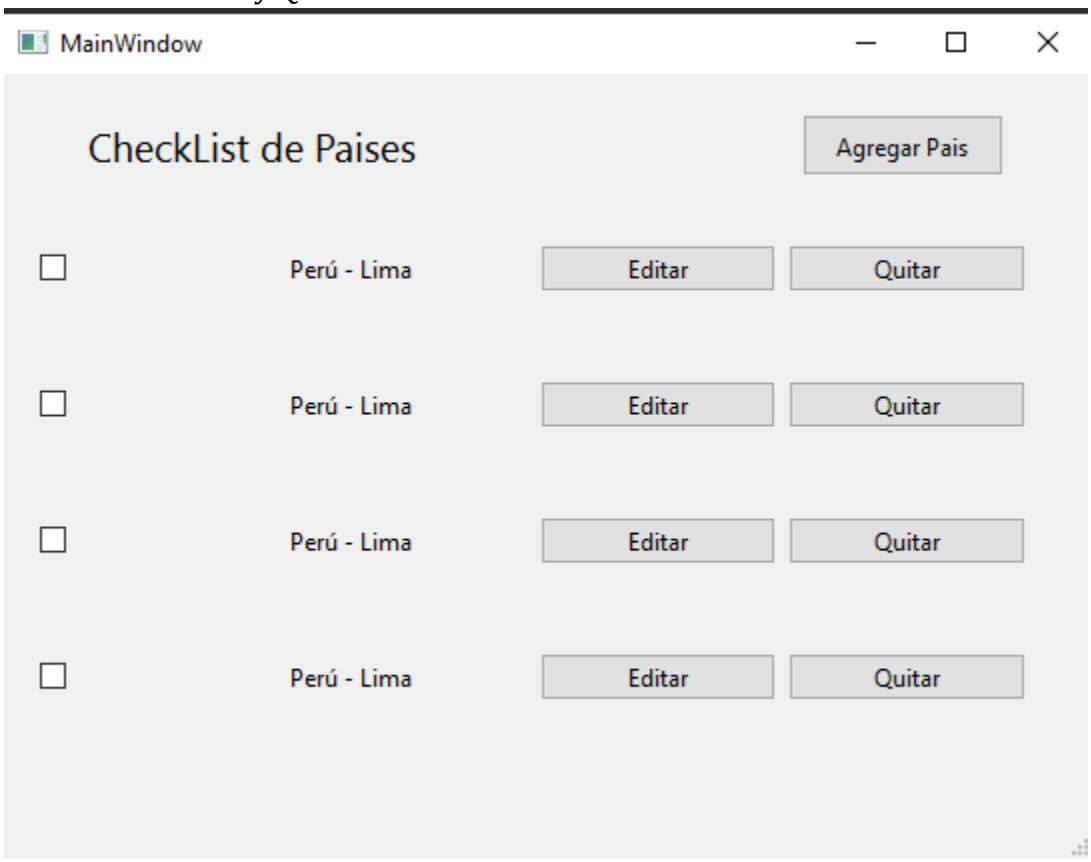
Usar el ejemplo anterior Qt/Widgest (parte 2) para implementar la interface gráfica que permita

a) El botón “Agregar tarea” debe cambiar por “Agregar país”

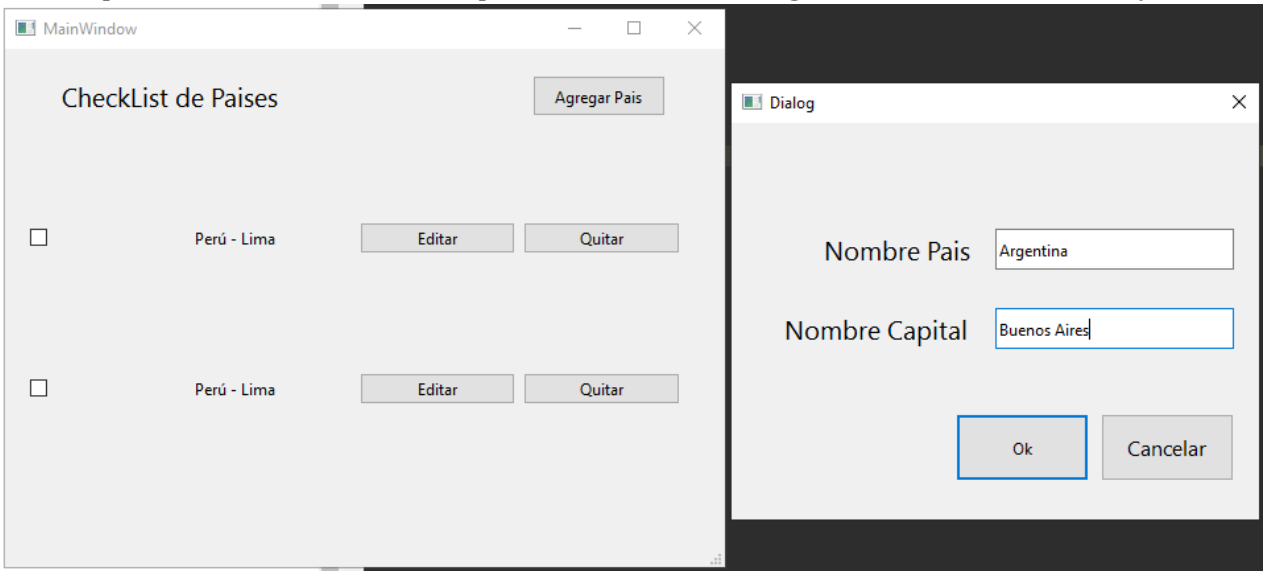


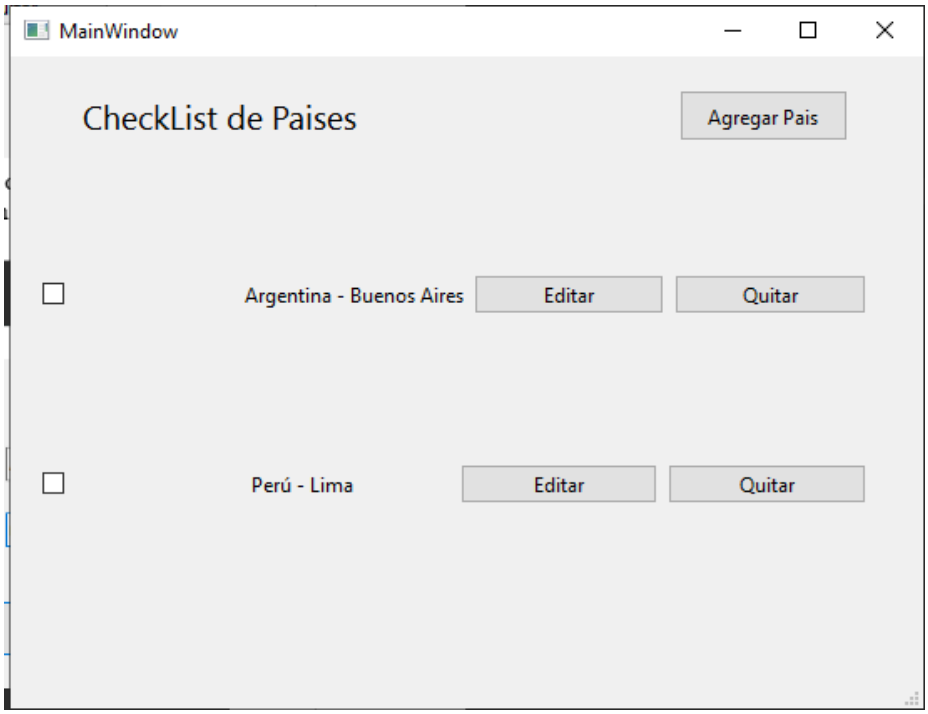
b) Cuando se hace clic en “Agregar país”, se debe mostrar el checkbox, un

objeto Texto con el nombre de un País y nombre de su capital (Perú – Lima) y los botones Editar y Quitar.

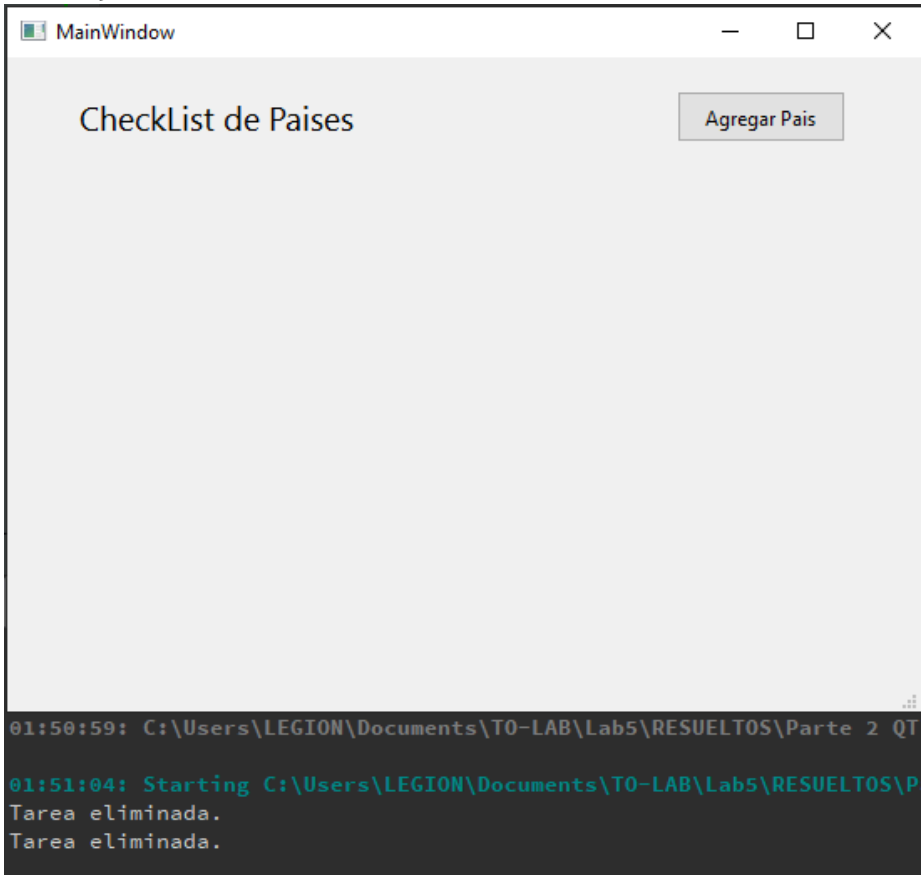


c) El botón Editar debe permitir cambiar los nombres de País y/o capital (ya que el usuario debe revisar que los checklist no tengan la misma información).





d) El botón Quitar debe eliminar toda la fila de objetos (tal cual como viene funcionando).



Código:

Pais.h

```
C/C++
#pragma once

#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui { class Pais; }
QT_END_NAMESPACE

class Pais : public QWidget {
    Q_OBJECT

public:
    explicit Pais(QWidget *parent = nullptr);
    ~Pais();

signals:
    void removed(Pais* pais);

private slots:
    void editPais();

private:
    Ui::Pais *ui;
    QString pais;
    QString capital;
};
```

EditPaisDialog.h

```
C/C++
#pragma once

#include <QDialog>

namespace Ui {
class EditPaisDialog;
}

class EditPaisDialog : public QDialog {
    Q_OBJECT

public:
    explicit EditPaisDialog(QWidget *parent = nullptr);
    ~EditPaisDialog();
```

```
    void setPaisInfo(const QString &pais, const QString &capital);
    QString getPais() const;
    QString getCapital() const;

private:
    Ui::EditPaisDialog *ui;
};
```

MainWindow.h

```
C/C++
#pragma once

#include <QMainWindow>
#include <QVector>
#include "Pais.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

public slots:
    void addPais();
    void removePais(Pais* pais);

private:
    Ui::MainWindow *ui;
    QVector<Pais*> paises;
};
```

Pais.cpp

```
C/C++
#include "Pais.h"
#include "ui_Pais.h"
#include "EditPaisDialog.h"
#include <QPushButton>
#include <QLabel>
#include <QCheckBox>
```

```
Pais::Pais(QWidget *parent) :
    QWidget(parent), ui(new Ui::Pais), pais("Perú"), capital("Lima") {
    ui->setupUi(this);

    ui->labelPais->setText(pais + " - " + capital);

    connect(ui->removeButton, &QPushButton::clicked, this, [this]() {
        emit removed(this);
    });

    connect(ui->editButton, &QPushButton::clicked, this, &Pais::editPais);

    setMinimumHeight(50);
    setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
}

Pais::~Pais() {
    delete ui;
}

void Pais::editPais() {
    EditPaisDialog dialog(this);
    dialog.setPaisInfo(pais, capital);
    if (dialog.exec() == QDialog::Accepted) {
        pais = dialog.getPais();
        capital = dialog.getCapital();
        ui->labelPais->setText(pais + " - " + capital);
    }
}
```

EditPaisDialog.cpp

```
C/C++
#include "EditPaisDialog.h"
#include "ui_EditPaisDialog.h"

EditPaisDialog::EditPaisDialog(QWidget *parent) :
    QDialog(parent), ui(new Ui::EditPaisDialog) {
    ui->setupUi(this);

    connect(ui->okButton, &QPushButton::clicked, this, &QDialog::accept);
    connect(ui->cancelarButton, &QPushButton::clicked, this, &QDialog::reject);
}

EditPaisDialog::~EditPaisDialog() {
    delete ui;
}

void EditPaisDialog::setPaisInfo(const QString &pais, const QString &capital) {
```

```
    ui->paisLineEdit->setText(pais);
    ui->capitalLineEdit->setText(capital);
}

QString EditPaisDialog::getPais() const {
    return ui->paisLineEdit->text();
}

QString EditPaisDialog::getCapital() const {
    return ui->capitalLineEdit->text();
}
```

MainWindow.cpp

```
C/C++
#include "MainWindow.h"
#include "ui_MainWindow.h"
#include "Pais.h"
#include <QVBoxLayout>
#include <QDebug>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow), paises() {
    ui->setupUi(this);

    connect(ui->addPaisButton, &QPushButton::clicked, this,
&MainWindow::addPais);
}

MainWindow::~MainWindow() {
    delete ui;
}

void MainWindow::addPais() {
    Pais *pais = new Pais(this);
    ui->paisLayout->addWidget(pais);
    paises.append(pais);
    connect(pais, &Pais::removed, this, &MainWindow::removePais);
    ui->paisLayout->update();
}

void MainWindow::removePais(Pais* pais) {
    ui->paisLayout->removeWidget(pais);
    paises.removeOne(pais);
```

```
    pais->deleteLater();
    qDebug() << "Tarea eliminada.";
}
```

5. Cuestionario

a. Revisar y encontrar diferencias, ventajas y desventajas entre plantillas y polimorfismo.

Diferencias

Plantillas (Templates):

- Las plantillas permiten escribir código genérico que puede funcionar con cualquier tipo de datos.
- Son una característica de tiempo de compilación. Esto significa que el tipo específico se resuelve en el momento de la compilación, y se genera una versión separada de la función o clase para cada tipo de dato usado.
- No requiere herencia ni clases base.

Polimorfismo:

- El polimorfismo permite que un objeto de una clase derivada se maneje a través de un puntero o referencia a su clase base.
- Es una característica de tiempo de ejecución, especialmente cuando se utiliza con herencia y métodos virtuales.
- Requiere que exista una relación de herencia entre las clases (clase base y derivada).

Ventajas y desventajas

Aspecto	Plantillas	Polimorfismo
Ventaja	Permiten código genérico y optimización en tiempo de compilación.	Permiten flexibilidad y extensibilidad en tiempo de ejecución.
Desventaja	Generan código duplicado para cada tipo de dato utilizado, aumentando el tamaño del ejecutable.	Menor rendimiento debido a la sobrecarga de llamadas virtuales.
Tiempo de ejecución	Resuelto en tiempo de compilación (más rápido).	Resuelto en tiempo de ejecución (más flexible).
Relación de clases	No es necesario una relación jerárquica entre clases.	Requiere una jerarquía de herencia.

Uso recomendado	Situaciones en las que se necesita alto rendimiento con distintos tipos de datos.	Situaciones que requieren una flexibilidad en el tiempo de ejecución.

b. Revisar sobre funciones amigas, cómo se integra a las plantillas.

Una función amiga es una función que no es un miembro de una clase, pero tiene acceso a sus miembros privados y protegidos. Esto permite que funciones externas interactúen con la clase sin formar parte de ella directamente. Las funciones amigas se declaran dentro de la clase con la palabra clave friend.

Pueden integrarse con plantillas para que funciones genéricas tengan acceso directo a los miembros de una clase específica. Existen dos maneras comunes de usar funciones amigas con plantillas

- Función amiga para todas las instancias de la clase plantilla:

```C/C++
template <typename T>
class MiClase {
     friend void funcionAmiga(const MiClase& obj); // Amigo de todas las instancias de MiClase
    T dato;
};
```

Esto permite que una función sea amiga de todas las instancias de la plantilla, independientemente del tipo.

- Función amiga específica para cada instancia de la plantilla:

```C/C++
template <typename T>
class MiClase {
      friend void funcionAmiga(const MiClase<T>& obj) { // Amigo solo de la instancia específica de T
        // Accede a los miembros privados de MiClase<T>
    }
    T dato;
};
```

Esto crea una función amiga específica para cada tipo de dato utilizado en la plantilla, lo cual es más común y flexible.

Algunas ventajas de usar funciones amigas son que permiten acceder a miembros privados de clases sin necesidad de hacerlos públicos, lo que facilita

la encapsulación y modularización también son útiles cuando se necesita interacción entre clases plantillas o entre una función y una clase plantilla específica.

c. Diferencias, ventajas y desventajas de las clases QVector y QList.

Aspecto	QVector	QList
Almacenamiento	Almacena los elementos en un bloque contiguo de memoria, similar a std::vector.	En versiones antiguas de Qt, QList podía usar una lista doblemente enlazada internamente. En versiones modernas (Qt 5 en adelante), se aproxima a un std::vector.
Acceso aleatorio	Muy rápido debido a su almacenamiento contiguo. Ideal para acceder a elementos por índice.	Rápido, aunque generalmente un poco más lento que QVector para acceso aleatorio.
Inserciones/borrado en el medio	Menos eficiente, ya que mover elementos en un bloque contiguo implica realocar y copiar datos.	En versiones anteriores de Qt, QList era más flexible para insertar y borrar en el medio, pero en las versiones actuales este comportamiento se ha unificado.
Rendimiento en tiempo de ejecución	Más predecible y consistente en rendimiento, ideal para operaciones de alto rendimiento con datos grandes.	Menos consistente en rendimiento para datos grandes, aunque su desempeño puede variar dependiendo de la versión de Qt y del tipo de datos almacenado.
Uso recomendado	Datos grandes o cuando se necesita acceso rápido mediante índices.	Datos pequeños (menos de 16 bytes) o cuando se almacenan punteros, donde QList es más eficiente debido a su optimización en el manejo de objetos pequeños.
Sobrecarga de memoria	Menor sobrecarga de memoria en comparación con QList, especialmente para tipos de datos más grandes.	QList puede tener una mayor sobrecarga de memoria para ciertos tipos de datos, aunque maneja eficientemente tipos pequeños y punteros.
Flexibilidad de uso	Ideal para estructuras que no cambian mucho de tamaño, ya que realocar datos grandes es	Más adecuado para situaciones que requieren una estructura de datos con inserciones y eliminaciones

	costoso.	frecuentes.
Desventajas	- Menor flexibilidad para inserciones en el medio. - Realocar datos grandes puede ser costoso en términos de rendimiento.	- En versiones antiguas de Qt, QList podía ser menos eficiente en el manejo de datos grandes debido a la sobrecarga de memoria y su estructura interna variable.

6. Conclusiones:

En el desarrollo de aplicaciones con Qt, hemos explorado la implementación de clases y funciones, destacando el uso de plantillas y funciones amigas para facilitar la interacción entre elementos de la interfaz y la lógica del programa. Además, abordamos la creación de la clase Pais, donde configuramos elementos de la interfaz como botones y conectores, asegurando una funcionalidad adecuada y la correcta gestión de errores de compilación. También analizamos los contenedores QVector y QList, identificando sus ventajas y desventajas; QVector se destacó por su rendimiento en el manejo de grandes volúmenes de datos, mientras que QList demostró ser más eficiente para datos pequeños y punteros. Este enfoque integral permite desarrollar aplicaciones efectivas y optimizadas en Qt, adaptadas a las necesidades específicas del proyecto.

7. Referencias:

[1] J. A. Meyer and M. M. Koss, "C++ Templates: The Complete Guide," 2nd ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2006.
[2] B. Stroustrup, "The C++ Programming Language," 4th ed. Boston, MA, USA: Addison-Wesley, 2013.
[3] J. W. McCormack, "Polymorphism in C++: An Overview," Journal of Computer Science and Technology, vol. 29, no. 3, pp. 471-482, 2014.
[4] Qt Documentation, "Qt Core Module," [Online]. Available: https://doc.qt.io/qt-5/qtcore-index.html [Accessed: 28-Oct-2024].
[5] Qt Documentation, "Containers and Data Structures," [Online]. Available: https://doc.qt.io/qt-5/qtcore-containers.html [Accessed: 28-Oct-2024].
[6] Qt Documentation, "QVector Class," [Online]. Available: https://doc.qt.io/qt-5/qvector.html [Accessed: 27-Oct-2024].
[7] Qt Documentation, "QList Class," [Online]. Available: https://doc.qt.io/qt-5/qlist.html [Accessed: 29-Oct-2024].
[8] C++ reference, online: https://en.cppreference.com/w/
[9] Qt, [Online]. Available: https://www.qt.io/
[10] QT Creator Manual.
[11] Ceballos, F.J., "Programación Orientada a Objetos con C++," 2da ed., 1997.
[12] Robert Lafore, "Object-Oriented Programming in C++," 4th edition.