

LICENCIATURA EN AUTOMATIZACION Y CONTROL

PROGRAMACION

Docentes

Santiago Di Marco
Andres Singls

Alumno

Cristian Gonzalo Vera

Legajo

420581

Proyecto

TP#1- Calculadora en QT

INDICE

1. Resumen introductorio

- 1.1. Contexto de aprendizaje
- 1.2. Objetivos del trabajo práctico
- 1.3. Herramientas utilizadas

2. Consigna original

Enunciado transcripto del TP

3. Fundamentos teóricos

- 3.1. ¿Qué es Qt?
- 3.2. ¿Qué es un archivo .ui?
- 3.3. ¿Qué es una señal (signal) en Qt?
- 3.4. ¿Qué es un slot en Qt?
- 3.5. ¿Qué es la variable *ui y cómo se usa?

4. Desarrollo e implementación

- 4.1. Análisis del problema y planteo algorítmico
- 4.2. Construcción de la interfaz gráfica
- 4.3. Separación entre lógica y vista
- 4.4. Declaración y uso de variables auxiliares
- 4.5. Diseño del slot principal manejarEntrada()
- 4.6. Funciones auxiliares implementadas
- 4.7. Manejo de errores y validaciones
- 4.8. Pruebas, casos y validación final

5. Estructura del repositorio de soporte

- 5.1. Descripción de carpetas y archivos
- 5.2. Instrucciones para compilar y ejecutar

6. Conclusiones personales

- 6.1. Reflexión sobre el uso de Qt en el contexto de la carrera
- 6.2. Dificultades encontradas y cómo se resolvieron
- 6.3. Posibles mejoras futuras

1. Resumen introductorio

1.1. Contexto de aprendizaje

Este trabajo práctico se enmarca en la asignatura de *Programación* correspondiente a la **Licenciatura en Automatización y Control**, donde se busca que el estudiante incorpore competencias en el desarrollo de interfaces gráficas de usuario (GUI) aplicando el framework **Qt** en conjunto con el lenguaje **C++**.

A través del desarrollo de una **calculadora funcional**, se fomenta la integración de conocimientos de lógica algorítmica, estructura de datos, manejo de eventos, orientación a objetos y diseño modular de software. Además, se entrena al estudiante en la construcción de proyectos utilizando una metodología clara de separación entre lógica y presentación.

Este tipo de proyectos actúa como puente entre los entornos visuales de desarrollo modernos y el trabajo con estructuras fundamentales del lenguaje C++, articulando teoría y práctica en una solución funcional.

1.2. Objetivos del trabajo práctico

- Diseñar y desarrollar una **aplicación de escritorio con interfaz gráfica** usando Qt.
 - Implementar el modelo de señales y slots de Qt para la interacción entre usuario y programa.
 - Integrar y aplicar conceptos como:
 - Variables auxiliares.
 - Evaluación de expresiones.
 - Manejo de errores.
 - Separar correctamente la **vista (UI)** de la **lógica del programa** mediante clases y funciones bien organizadas.
 - Validar el funcionamiento del programa a través de **pruebas controladas** y depuración manual.
-

1.3. Herramientas utilizadas

- **Qt Creator** (Entorno de desarrollo visual para proyectos Qt con soporte para archivos .ui, CMake y depuración)
 - **Qt Framework 6.x** (Bibliotecas para interfaces gráficas, slots, eventos y layouts)
 - **Lenguaje C++** (Base del desarrollo lógico del proyecto)
 - **Sistema operativo Windows 10**
 - **Control de versiones Git** (opcional para seguimiento y entrega)
 - **Captura de video y edición** (para presentación final)
-

2. Consigna original

Crear una calculadora simple de números utilizando Qt que pueda:

- Realizar operaciones básicas como suma, resta, multiplicación y división.
- La calculadora deberá tener una interfaz gráfica compuesta por botones para:
 - Los dígitos del 0 al 9
 - Para los operadores (+, -, *, /)
 - Para mostrar el resultado
 - Para borrar todo lo que se ingresó hasta el momento

Además, debe haber una etiqueta para mostrar los números ingresados, la operación y el resultado de la operación.

Todos los botones deben ser conectados a sus slots todo mediante programación con la función connect de QT.

3. Fundamentos teóricos

3.1. ¿Qué es Qt?

Qt es un framework de desarrollo multiplataforma ampliamente utilizado para la creación de interfaces gráficas de usuario (GUI), así como también para aplicaciones que requieren acceso a redes, bases de datos, archivos XML, multimedia y más. Está

escrito en C++ y ofrece herramientas poderosas como Qt Designer, que facilita el diseño visual de interfaces.

En el marco de este trabajo práctico, utilicé Qt para construir una calculadora gráfica que responde a eventos del usuario, empleando su sistema de señales y slots para lograr una interacción fluida entre los botones y la lógica de cálculo.

3.2. ¿Qué es un archivo .ui?

Un archivo .ui es un archivo XML generado por Qt Designer que describe la estructura visual de una interfaz gráfica. Contiene información sobre los widgets, su jerarquía, disposiciones (layouts), propiedades y conexiones.

En este proyecto, el archivo mainwindow.ui representa visualmente la ventana principal de la calculadora. Aunque este archivo no se edita manualmente, es procesado por el compilador mediante la herramienta uic (User Interface Compiler), que lo transforma en código C++ accesible desde la clase principal a través del puntero *ui.

3.3. ¿Qué es una señal (signal) en Qt?

En Qt, una **señal** es un mecanismo que informa cuando ocurre un evento. Por ejemplo, al presionar un botón, se emite la señal clicked(). Esta señal no contiene lógica propia, sino que su propósito es comunicar que "algo ocurrió".

En el desarrollo de la calculadora, cada botón de la interfaz emite su señal clicked() al ser presionado. A través de la función connect(), estas señales fueron redirigidas a un único método central que se encarga de interpretar el texto del botón y ejecutar la acción correspondiente.

3.4. ¿Qué es un slot en Qt?

Un **slot** es una función que se ejecuta en respuesta a una señal. Qt permite definir slots personalizados para manejar eventos específicos.

En este proyecto, implementé un único slot público llamado manejarEntrada(), que se encarga de procesar todos los clics realizados sobre los botones de la calculadora. Este enfoque centralizado me permitió simplificar la lógica del programa, evitando múltiples funciones separadas para cada botón.

3.5. ¿Qué es la variable *ui y cómo se usa?

La variable *ui es un puntero que apunta a una instancia de la clase generada a partir del archivo .ui. Esta clase contiene todos los widgets (botones, campos de texto, layouts) que componen la interfaz gráfica.

Desde el código fuente de MainWindow, accedo a los elementos visuales utilizando este puntero, por ejemplo ui->display o ui->btn5. Gracias a esto, puedo leer y modificar el contenido de los elementos, conectar señales, y actualizar dinámicamente la interfaz según la lógica implementada.

4. Desarrollo e implementación

4.1. Análisis del problema y planteo algorítmico

El problema planteado consiste en desarrollar una calculadora de operaciones básicas (suma, resta, multiplicación y división) con interfaz gráfica utilizando Qt. La calculadora debía operar con números decimales y soportar operaciones secuenciales simples.

Inicialmente, se pensó el funcionamiento en términos de eventos y estados. Toda interacción se reduce a una secuencia de clics: primero uno o más dígitos, luego un operador, luego otro dígito, y finalmente el botón igual para obtener un resultado. También se identificaron acciones de control como **borrar (DEL)**, **reiniciar (SUP/AC)** y **validación de errores**.

Algoritmo general propuesto:

1. Inicializar la interfaz y las variables.
2. Mientras no haya error:
 - Si se presiona un dígito → concatenarlo al número actual.
 - Si se presiona un operador:
 - Guardar el primer operando.
 - Guardar el operador.
 - Prepararse para ingresar el segundo operando.
 - Si se presiona "=":
 - Evaluar la operación con los dos operandos.
 - Mostrar resultado o error.
 - Si se presiona DEL → eliminar último carácter del número.
 - Si se presiona SUP/AC → reiniciar todo.

3. Si ocurre una entrada inválida o división por cero → mostrar mensaje y bloquear operación.

Este modelo se traduce luego en una implementación basada en un único método central de entrada, varias funciones auxiliares, y una clara distinción entre lógica y presentación.

4.2. Construcción de la interfaz gráfica

La interfaz fue construida utilizando Qt Designer a través del archivo `mainwindow.ui`. Se utilizó un `QVBoxLayout` principal que contiene dos componentes:

- **Un `QLineEdit (display)`:** utilizado como pantalla para mostrar los números y mensajes.
- **Un `QGridLayout (gridLayout)`:** donde se ubican los botones numéricos y de operaciones.

Los botones se agregaron manualmente desde el toolbox de Qt y se les asignó nombres representativos (`btn0`, `btnSuma`, `btnIgual`, etc.). Luego, todos los botones fueron conectados **por código** al mismo slot mediante la función `connect()`.

Se utilizó el sistema de layouts para que los botones se ajusten automáticamente al tamaño de la ventana. Para el botón “=”, se configuró una ocupación de dos filas en el grid.

4.3. Separación entre lógica y vista

Una decisión clave del diseño fue **mantener una clara separación entre la lógica de la calculadora y su presentación visual**. Esto se logró al seguir una organización como la siguiente:

- Todo lo referente a **cómo se ve** la calculadora (botones, layouts, display) está contenido en `mainwindow.ui`.
- La **lógica del funcionamiento** (procesamiento de la entrada, operaciones, validaciones) se desarrolla exclusivamente en `mainwindow.cpp`.

Además, se evitó el uso de múltiples slots individuales, centralizando toda la lógica de interacción en un único slot: `manejarEntrada()`, lo cual permite tener una visión integral del flujo de ejecución y facilita el mantenimiento del código.

4.4. Declaración y uso de variables auxiliares

Para controlar el estado de la calculadora, se declararon las siguientes variables en la clase MainWindow, con funciones bien definidas:

- QString numeroActual: guarda lo que el usuario está escribiendo.
- QString operadorActual: almacena el operador ingresado (+, -, ×, ÷).
- double operando1: primer número ingresado antes del operador.
- bool esperandoSegundoOperando: marca si el sistema espera el segundo número.
- bool resultadoMostrado: indica si se acaba de mostrar un resultado.
- QString mensajeError y bool errorDetectado: se usan para capturar y manejar errores como división por cero o formatos inválidos.

Estas variables permiten mantener un **estado consistente** durante el uso de la aplicación y son manipuladas únicamente desde la lógica de control, sin interferir directamente con los elementos gráficos.

4.5. Diseño del slot principal manejarEntrada()

El corazón de la lógica de la calculadora es el método manejarEntrada(), un **slot público** que centraliza el tratamiento de todos los botones. Fue diseñado para que todos los clics se procesen a través de este único punto de entrada.

Estrategia de funcionamiento:

1. Se identifica el botón presionado usando sender() y se recupera su texto.
2. Según el contenido del botón se realiza una de las siguientes acciones:
 - Si es un dígito → se concatena al número en construcción.
 - Si es un operador → se guarda el primer operando y el operador.
 - Si es "=" → se realiza la operación.
 - Si es "DEL" o "SUP" → se borra un carácter o se reinicia todo.
 - Si es un punto decimal → se agrega si no estaba ya presente.

- Si el texto no es reconocido → se lanza un mensaje de error.

Esta función contiene validaciones como evitar puntos dobles, entrada incompleta, o varios operadores consecutivos, y deja preparados los estados necesarios para continuar la operación.

4.6. Funciones auxiliares implementadas

Para mantener limpio y entendible el código, se encapsularon comportamientos repetitivos o específicos en funciones auxiliares:

- limpiarTodo(): reinicia el estado interno y limpia el display.
- borrarUltimoCaracter(): elimina el último carácter del número en curso.
- mostrarError(QString): despliega un mensaje de advertencia y marca el sistema como bloqueado.
- esOperador(QString): determina si un texto representa un operador aritmético válido.
- calcularResultado(): toma operando1, operadorActual, numeroActual y evalúa la operación, mostrando el resultado o indicando errores.

Esta modularidad facilita la comprensión, la depuración y el reuso de componentes si se expandiera la funcionalidad de la calculadora.

4.7. Manejo de errores y validaciones


Durante el desarrollo se le dio mucha importancia al **tratamiento de entradas erróneas**. La lógica de validación está presente en varios niveles:

- **Antes de evaluar una operación**, se verifica que ambos operandos estén presentes.
- **División por cero** genera un mensaje específico y bloquea el funcionamiento.
- **Entrada de operadores sin operandos** o uso de caracteres no reconocidos dispara errores controlados.

Una vez que ocurre un error, la variable errorDetectado se activa. Esto impide cualquier operación posterior hasta que el usuario presione “SUP” o “AC”, que reinicia el sistema. Este control evita resultados incoherentes o crashes.

4.8. Pruebas, casos y validación final

Luego de implementar la lógica y la interfaz, se realizaron **pruebas funcionales** directamente sobre el ejecutable generado. Algunos de los casos probados fueron:

- Ingresos básicos como $7 + 3 = \rightarrow 10$
- Operaciones con decimales $\rightarrow 2.5 \times 4 = \rightarrow 10$
- División válida $\rightarrow 9 \div 3 = \rightarrow 3$
- División por cero $\rightarrow 9 \div 0 = \rightarrow$  No se puede dividir por cero
- Secuencias inválidas \rightarrow solo “+”, solo “=”, múltiples “.”, etc.

Se detectó y corrigió un **problema de bloqueo** que se producía cuando la calculadora entraba en estado de error y no respondía a los botones de control. Este comportamiento se ajustó permitiendo que “SUP” y “DEL” funcionen incluso tras errores.

5. Estructura del repositorio de soporte

Con el objetivo de mantener una documentación clara, ordenada y reutilizable, se organizó un repositorio estructurado por bloques funcionales. Cada carpeta cumple una función específica dentro del proceso de desarrollo, permitiendo rastrear desde la consigna inicial hasta la solución final y su presentación. A continuación se describe el contenido principal:

5.1. Descripción de carpetas y archivos principales

- **A_requisitos/**
Esta carpeta contiene el enunciado original del trabajo práctico, tal como fue entregado por la cátedra. Se mantiene sin modificaciones como referencia directa para verificar los objetivos propuestos.
- **B_teoría/**
Se reúnen en esta sección los materiales de estudio y consulta utilizados para resolver el trabajo. Incluye manuales de Qt, apuntes sobre señales y slots, y documentos relacionados al uso de interfaces .ui y la separación lógica en C++.
- **C_solucion/**
Es el núcleo del proyecto. Contiene la carpeta CalculadoraQt/ que incluye

todos los archivos fuente del proyecto Qt desarrollado, tales como mainwindow.ui, mainwindow.cpp, mainwindow.h, main.cpp y el archivo CMakeLists.txt para compilar el proyecto en cualquier entorno compatible. La solución se encuentra completamente funcional.

- **D_presentacion/**

En esta sección se incorporó un video demostrativo donde se visualiza el funcionamiento de la calculadora. Es una síntesis audiovisual que puede ser utilizada como presentación rápida.

- **README.md**

Archivo principal del repositorio. Actúa como guía estructurada con índice, instrucciones de compilación, descripción del proyecto, reflexiones y referencias cruzadas a los materiales utilizados. Resume todo el desarrollo e implementación.

- **LICENSE**

Archivo que define la licencia del proyecto.

5.2. Instrucciones para compilar y ejecutar

Para ejecutar correctamente la calculadora en Qt, se deben seguir los siguientes pasos:

✂ Requisitos:

- Qt Creator (recomendado: versión 6.x)
- Compilador compatible con C++17 o superior
- Sistema operativo: Windows, Linux o macOS

Pasos:

1. Clonar o descargar el repositorio desde GitHub:

git clone https://github.com/Automatizacion-y-Control/TP1_programacion_LAYC.git

2. Abrir Qt Creator y cargar el archivo CMakeLists.txt ubicado dentro de C_solucion/CalculadoraQt.
3. Verificar que el kit de compilación esté correctamente configurado (MSVC, MinGW o GCC según sistema).
4. Compilar el proyecto desde el botón **Build** o menú **Build → Build Project**.
5. Ejecutar la aplicación desde el botón **Run** o tecla Ctrl+R.

6. Conclusiones personales

6.1. Reflexión sobre el uso de Qt en el contexto de la carrera

Este trabajo representó mi primer contacto formal con Qt, y debo reconocer que fue una grata sorpresa. Me encontré con un entorno robusto, moderno y extremadamente útil, que combina la potencia del lenguaje C++ con una estructura visual amigable y eficiente para el desarrollo de interfaces gráficas.

Ya tenía experiencia previa en programación en C, especialmente aplicada a microcontroladores (PIC con XC8, ESP-IDF de Espressif, entre otros), por lo que la curva de aprendizaje no fue compleja. Qt, sin embargo, me abrió una nueva dimensión de posibilidades, especialmente para aplicaciones que requieren interacción visual con el usuario, manteniendo al mismo tiempo control total sobre la lógica del programa.

Considero que integrar Qt en la formación de la Licenciatura en Automatización y Control es altamente pertinente, ya que se trata de una herramienta profesional, ampliamente usada en la industria para el diseño de HMI, paneles de control, software de instrumentación, simuladores y prototipos funcionales.

6.2. Dificultades encontradas y cómo se resolvieron

Una de las principales dificultades técnicas que enfrenté fue la desactualización de gran parte del material de referencia que circula en la web. Muchos ejemplos de código estaban basados en versiones anteriores de Qt, lo cual generó inconsistencias, especialmente con funciones que ya han sido modificadas o deprecated, como el manejo de `connect()` o ciertas librerías específicas.

Esto me obligó a contrastar constantemente con la documentación oficial de Qt, que finalmente fue el recurso más confiable. A partir de ahí, logré resolver las incompatibilidades, corregir las llamadas incorrectas y adaptar las soluciones al entorno actual que estoy utilizando. Esta experiencia me reafirmó la importancia de trabajar siempre con documentación oficial y actualizada.

6.3. Posibles mejoras futuras

No tengo planeado continuar el desarrollo de esta calculadora en particular, pero reconozco que ofrece un excelente punto de partida para múltiples ampliaciones. Si tuviese más tiempo, exploraría:

- La visualización de operaciones completas (expresiones encadenadas).
- La transformación a una calculadora científica.
- La incorporación de gráficos.
- El soporte para teclado físico.
- El desarrollo de una interfaz tipo display con apariencia de segmentos, algo que ya comencé a explorar mediante fuentes personalizadas.

También considero que esta base puede extenderse fácilmente a otros tipos de aplicaciones que requieren botones y display, como simuladores de controladores, teclados numéricos para acceso industrial, o incluso paneles interactivos de máquinas.