Sistemas informáticos
Curso 2005-2006

Fulltick Editor: Editor Gráfico de Interfaces Gráficas En C++

Alberto Antonio Del Barrio García
Francisco Javier Rincón Vallejos
Julio Sevillano Camarero

Dirigido por:
Profesora: Ana Gil Luezas
Departamento: Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Resumen

Este proyecto consiste en un editor gráfico de interfaces gráficas que genera un código C++ automáticamente asociado a la interfaz gráfica desarrollada por el usuario de forma que este código sea claro, limpio, inteligible y orientado a objetos lo que facilita que el usuario pueda relacionar ese código con los elementos de la interfaz, para si lo desea poder añadir código manualmente y seguir desarrollando su aplicación.

También pretendemos que sea una solución a las actuales existentes, ya que éstas o son de pago, o carecen de editores de interfaces gráficas o son difíciles de programar o generan un código sucio y no orientado a objetos. Es más, muchas de ellas solo están preparadas para trabajar en entorno Windows, mientras que nuestro proyecto permite trabajar tanto en Windows como en Linux.

Para nuestro editor nos hemos basado en la librería FLTK (Fast Light ToolKit) que es una librería de código libre bajo la licencia LGPL (GNU Library Public License, Version 2), lo que nos ha permitido modificar o reutilizar el código existente en la misma para nuestros fines. FLTK es un conjunto de herramientas de interfaz gráfico en C++ para UNIX®/Linux® (X11), Microsoft® Windows®, and MacOS® X. FLTK es un moderno interfaz gráfico que soporta gráficos 3D mediante OpenGL.

Palabras Clave: Interfaz Gráfico (GUI), Editor gráfico, C++, FLTK, Widget, Evento, Callback.

Abstract

This project is a graphical interfaces editor which automatically generates C++ code associated to a graphical interface developed by the user, so this code is clear, clean, intelligible and Object-Oriented (OO). This editor facilitates the user to relate that code to the elements of the interface, add code manually and continue developing his application, as he desires.

We also try our project is a solution to the current existing ones, since these either are of payment, or lack graphical interfaces editors or are difficult to develop or generate a dirty and not OO code. Even more, many of them are just prepared to work in Windows environment, whereas our project allows to be employed both at Windows and Linux.

In order to develop our project we have worked with FLTK, an open source library under the terms of LGPL (GNU Library Public License, Version 2) what has allowed us to modify or reuse the existing code in the same one for our aims. FLTK is a cross-platform C++ GUI toolkit for UNIX®/Linux® (X11), Microsoft® Windows®, and MacOS® X. FLTK provides modern GUI functionality without the bloat and supports 3D graphics via OpenGL®.

Keywords: Graphical User Interface (GUI), Graphical Editor, C++, FLTK, Widget, Event, Callback.

1. Introducción

Hoy en día cualquier programador que intente diseñar interfaces gráficas en C++ se encuentra con que no existen soluciones fáciles de usar de forma que pueda diseñar su interfaz gráfica y al mismo tiempo se genere el código asociado a la misma de forma que éste sea claro, comprensible y sencillo de entender. Todo esto se puede constatar en multitud de foros de Internet, donde la gente se queja de que no existen este tipo de herramientas.

En cuanto a las herramientas existentes en el mercado cabe destacar las más usadas por los usuarios como Visual C++ ó Borland C++ Builder, pero o bien carecen de editores de interfaces gráficas o son bastante difíciles de programar. Estas herramientas pueden trabajar exclusivamente bajo el entorno de Windows, por lo que muchos programadores que se decantan por utilizar otros sistemas operativos como Linux no disponen de ellas. Todo esto es una desventaja de C++ respecto a Java, ya que Java dispone del Swing, que permite realizar interfaces gráficas de forma fácil.

Otra alternativa es desarrollar dlls en Visual C++ para luego llamarlas desde Borland Delphi o Visual Basic. Esto tiene el inconveniente de utilizar entornos de programación distintos, los cuales el usuario debe conocer, y además tenemos el problema de Visual Basic que tiene unas limitaciones de tiempo muy grandes cuando se trata de grandes proyectos.

En cuanto al mundo del código libre existe una mayor variedad de opciones como pueden ser las librerías FLTK, GTK+, etc. Estas librerías están al alcance de cualquier usuario y puede programar interfaces gráficas con bastante facilidad, pero por otra parte tienen el inconveniente de que el código que se genera asociado a esa interfaz es muchas veces incomprensible y está bastante mal estructurado, por lo que el usuario que quiera entender ese código y/o modificar alguna parte del mismo debe realizar un estudio bastante detallado de las librerías.

1.1 Solución

Nuestro editor gráfico de interfaces gráficas pretende ser una solución a todos los anteriores problemas comentados, de forma que un usuario, incluso con bajos conocimientos de programación pueda diseñar una interfaz gráfica de un modo rápido y que genere automáticamente un código asociado a la interfaz que sea claro, limpio, inteligible y orientado a objetos de forma que el usuario pueda relacionar ese código con los elementos de la interfaz para si lo desea poder añadir código manualmente y seguir desarrollando su aplicación.

Después de un estudio detallado de varias librerías de código libre como son FLTK,GTK+ o FOX nos hemos decidido por usar para el desarrollo de nuestro proyecto la librería FLTK.

La librería FLTK (Fast Light ToolKit) es una librería de código libre bajo la licencia LGPL, lo que nos ha permitido modificar o reutilizar el código existente en la misma para nuestros fines. FLTK es un conjunto de herramientas de interfaz gráfico en C++ para X, OpenGL, y Windows.

1.1.1 ¿Por qué FLTK?

A continuación exponemos en qué se basan cada una de las librerías estudiadas:

- FOX: Es un toolkit escrito en C++, con un sistema de eventos basado en tablas, como por ejemplo lo están las MFC, o las OWL en Windows. Su principal peculiaridad es que el sistema de posicionamiento está basado en Layouts, como en Java. Es bastante completo y muy rápido. Esta es una librería reciente por lo que carece de la madurez de otras como FLTK. Además FLTK es más fácil de manejar y está mejor documentado. FOX sigue la licencia GNU Lesser General Public License (LGPL).
- GTK+: Hoy en día es probablemente el toolkit más usado en Linux, nació para desarrollar el GIMP (GNU Image Manipulation Program), que es una herramienta para retoque fotográfico y composición de imágenes, pero las buenas ideas se adoptan rápidamente. Técnicamente es un toolkit basado en layouts, como el AWT y el SWING (y como FOX), con un sistema de mensajes basado en slots y señales, tan sencillo como eficaz, está escrito en C, y para C, lo

que no le quita que haya bindings para C++, Python, Perl y algún lenguaje más, todos ellos de una enorme calidad, en concreto el de C++, llamado *GTKmm*, tiene un sistema de mensajes distinto, llamado libsig++, basado en templates. GTK+ tiene un sistema de temas que permite añadir degradados, texturas y gran cantidad de filigranas al interfaz. Hay un port de él bajo Windows, aunque no tiene toda su funcionalidad. Es además muy sencillo de aprender. Un problema que tiene es que hay que reservar y liberar memoria para cualquier objeto que haya que crear. GTK sigue la licencia GNU LGPL. Una pega que tiene esta librería es que su editor (llamado Glade) era de pago, lo que le resta bastantes puntos frente a FLTK. Además GTK+ utiliza un lenguaje no orientado a objetos (C), pero si utiliza orientación a objetos en todo el desarrollo (GTK+ es una librería orientada a objetos escrita en C), esto convierte al código fuente de GTK+ en bastante difícil de entender.

- Existen otras librerías como pueden ser Qt, X Toolkit, Motif, etc, pero estas no fueron objeto de estudio por nuestra parte, porque sus características no se adaptaban a nuestros intereses.

Después del estudio de las librerías anteriormente comentadas, finalmente nos decantamos por FLTK y estos son algunos de los motivos que nos llevaron a tomar esta decisión:

- FLTK es simple, incluso más simple que GTK.
- Soporta OpenGL 3D.
- Su ligereza permite enlazar estáticamente en los programas que lo usan, con un incremento mínimo del espacio que ocupan.
- Dispone de un editor de interfaces gráficos (Fluid), que no es muy vistoso, pero si eficiente. Un problema de este editor, es el código que genera, que no está ordenado y muchas veces no se entiende.
- Existe un buen tutorial para principiantes en PDF, lo que nos ha solucionado muchas dudas.
- El posicionamiento de controles es directo, aunque tiene un pequeño soporte para layouts.
- Su sistema de eventos se basa en funciones callback.
- Disponible para Windows y Linux, y es de código libre.

- Ofrece muy buen balance entre facilidad de uso y complejidad, y es bastante rápido.

1.2 Plataformas

El proyecto ha sido diseñado para las siguientes plataformas:

- Windows: En esta plataforma es donde hemos estado desarrollando nosotros, por lo que está demostrado el correcto funcionamiento del editor en este entorno. Cabe destacar que Windows es el sistema operativo más utilizado en el mundo, por lo que nuestro editor será accesible para una gran cantidad de usuarios.
- Linux: Tenemos una versión operativa aunque con ciertas limitaciones, ya que ha habido que cambiar bastantes cosas al migrar todo el trabajo a Linux. Esta migración ha sido realizada en paralelo con las últimas versiones del proyecto en Windows y hay algunas funcionalidades que no van del todo bien. También nos ha dado muchos problemas la librería Xerces, que ha sido muy difícil de integrar en el proyecto en Linux.

2. Herramientas utilizadas

2.1 FLTK

Como se ha comentado anteriormente FLTK es una libería C++ para interfaces gráficos para X, OpenGL, y Windows bajo la licencia LGPL.

A continuación explicamos como se realiza la instalación de esta librería tanto en Windows como en Linux, que han sido las plataformas en las cuales hemos comprobado su funcionamiento:

Instalación bajo Windows

Para la instalación de la librería FLTK seguir los siguientes pasos:

- Descargar la librería FLTK (versión 1.1.6) de la página www.fltk.org
- Descomprimir su contenido.
- Compilar el archivo fltkdll.dsp. Si se va a utilizar el Visual C++ o el C++ Builder este archivo se encuentra en la carpeta fltk-1.1.6\visualc. Si se va a utilizar el Visual Studio .Net el archivo se encuentra en la carpeta fltk-1.1.6\vcnet.
- Los ficheros de cabecera(.h) generados al compilar que se encuentran en la carpeta fltk-1.1.6\FL hay que copiarlos en la carpeta donde se instaló el Visual C++ en la ruta \Microsoft Visual Studio\VC98\include\. Hay que copiar la carpeta FL y su contenido.
- Copiar las bibliotecas estáticas (.lib) que se encuentran en fltk-1.1.6\lib en la carpeta lib del Visual C++, que se encuentra en \Microsoft Visual Studio\VC98\Lib.
- Copiar la dll generada en la carpeta fltk-1.1.6\visualc en la carpeta C:\WINDOWS\System.
- Cada vez que se vaya a diseñar un proyecto nuevo hacer:
  - En el Visual C++ ir a pestaña Project → Settings → Link.
  - Y añadir estas librerías: fltkd.lib kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib wsock32.lib fltkformsd.lib fltkgld.lib

fltkimagesd.lib fltkjpegd.lib fltkpngd.lib fltkzd.lib fltkdlld.lib msvcrtd.lib libcd.lib comctl32.lib

Instalación bajo Linux

Para la instalación de la librería FLTK seguir los siguientes pasos:

- Descargar la librería FLTK (versión 1.1.6) de la página www.fltk.org
- Descomprimir su contenido.
- Ejecutar ./configure, especificando, si se desea, el valor de ciertas variables de entorno que contienen rutas para guardar las librerías, las cabeceras, etc.
- Compilar con make.
- Instalar con make install.
- Nosotros hemos trabajado con Eclipse, y cada vez que se cree un proyecto nuevo hay que añadir las librerías libfltk.a, libforms.a, libimages.a y las librerías gráficas propias de Linux.

XERCES

Xerces es una herramienta que permite parsear ficheros XML y ver que estos ficheros son válidos. Está escrito en un subconjunto del lenguaje C++ que hace que sea portable a otras plataformas, esta librería facilita a las aplicaciones la lectura y escritura de ficheros XML. El Xerces ha sido utilizado en el módulo 3 a la hora de recorrer los ficheros XML generados por el módulo 2 para generar el código C++ y en el módulo 1, para abrir un proyecto previamente creado.

A continuación vamos a explicar cómo se instala esta librería tanto en Windows como en Linux:

Instalación bajo Windows

Para la instalación de la librería Xerces seguir los siguientes pasos:

- Descargar la librería Xerces (versión 2.7.0) de la página http://xml.apache.org/xerces-c/
- Descomprimir su contenido.

- Copiar en el directorio xercesc, contenido en el directorio include de la librería en la ruta donde se instaló el Microsoft Visual C++, dentro de la carpeta \Microsoft Visual Studio\VC98\include\.
- Copiar el fichero xerces-c_2D.lib, contenido en el directorio lib del Xerces en el directorio \Microsoft Visual Studio\VC98\Lib.
- Copiar el fichero xerces-c_2_7D.dll, contenido en el directorio bin de la librería en c:\Windows\system
- Al igual que en la intalación de la librería FLTK hay que entrar en la pestaña Project → Settings → Link y añadir la librería xerces-c_2D.lib cada vez que creemos un proyecto en el que queramos usar la librería Xerces.

Instalación bajo Linux

Para la instalación de la librería Xerces seguir los siguientes pasos:

- Descargar la librería Xerces (versión 2.7.0) de la página http://xml.apache.org/xerces-c/
- Descomprimir su contenido.
- Ejecutar la orden "export XERCESCROOT=<ruta completa de xerces-c-src2_7_0>"
- Necesitamos, para el siguiente paso tener instalada la herramienta autoconf, si no la tenemos podemos descargarla de la página http://www.gnu.org/software/autoconf/autoconf.html
- A continuación ejecutamos "cd $XERCESCROOT/src/xercesc" y luego "autoconf", lo que genera un script llamado "configure".
- Ejecutamos el script creado anteriormente con la orden "./runConfigure -plinux -cgcc -xg++ -minmem -nsocket -tnative –rpthread"
- Por último ejecutamos "make" y la libería estará lista para ser usada.
- Para poder usar el Xerces en un proyecto desde Eclipse tenemos que añadir las librerías xerces-c_2.o.27 y xerces-depdom_2.o.27.

3. Riesgos

Durante el desarrollo de todo proyecto se producen riesgos los cuales pueden tener consecuencias sobre el desarrollo del mismo, que pueden afectar de diversas formas según el impacto de dicho riesgo.

Para mitigar el daño causado por estos riesgos se realizó un estudio de los posibles riesgos que podríamos sufrir a lo largo del proyecto y cuáles serían las acciones que se realizarían para reducir el impacto de los mismos.

En este apartado mostramos este estudio realizado.

3.1 Identificación de Riesgos

- Riesgos tecnológicos:
  - Limitaciones en el laboratorio como pueden ser no disponer del software necesario, falta de horarios libres compatibles con los horarios de los miembros del grupo, falta de memoria o ancho de banda disponible, cortes del suministro eléctrico, etc.
  - Uso de diferentes versiones por parte de los miembros del grupo.

- Riesgos de personal:
  - Personal no cualificado para la actividad, es decir insuficiencia de conocimientos e inexperiencia. Hay que considerar que hay aspectos del proyecto que involucran conocimientos que aún no han sido incluidos en la formación de alguno de los componentes del grupo y que exigirán un trabajo extra.
  - Falta de entendimiento de los miembros del grupo. Las consecuencias serían graves en caso de producirse: un malentendido en un punto crucial del proyecto puede provocar que se desarrolle el mismo por dos vías distintas sin llegar a cumplir los requisitos esperados o añadiendo algunos adicionales sin planificación previa.

- Falta de determinados miembros del grupo por viajes ocasionales y dispersión en periodo vacacional debido a las distintas procedencias de los miembros.
- Falta de algún miembro del grupo por enfermedad. Las consecuencias son muy graves puesto que cada miembro es una pieza indispensable en la realización del proyecto.
- Abandono del proyecto por parte de uno o varios miembros del grupo.
- Falta de ánimo de trabajo en el grupo.
- Incompatibilidad de horarios considerables debido a las distintas asignaturas, trabajo, cursos, becas, etc.

- Riesgos de organización:
  - Que cambie la estructura del grupo por falta o por aumento de personal.
  - Que cambie el tutor del proyecto por algún motivo.
  - Mala coordinación de tareas que provoquen que un mismo trabajo sea realizado por varios miembros.
  - Pérdida de documentos o archivos.
  - Que no se respete la organización jerárquica de los componentes del grupo pudiendo realizar algún miembro del mismo tareas que no le correspondan sin el conocimiento del resto.
  - Desconocimiento por parte de algunos de los miembros del grupo de fechas de reuniones.

- Riesgos de requisitos:
  - Inestabilidad de requisitos. Aunque éstos se hayan dejado suficientemente claros en la documentación adjunta al proyecto, puede que algunos de ellos sean contradictorios o supongan tomar decisiones contrarias en algún punto de la realización del proyecto. Supondrían el retraso de la finalización del proyecto.
  - Cambio de los requisitos del producto antes de su entrega.
  - Incumplimiento de requisitos fijados por causa de falta de tiempo o desconocimiento de su forma de realización.

- Riesgos de estimación:

- No saber la fecha de entrega del proyecto con suficiente antelación.
- Conflictos de recursos temporales. Aunque los plazos se hayan establecido holgadamente pueden verse afectados por estos u otros riesgos que jamás había considerado el programador. Las consecuencias serían graves, como el retraso de la finalización del proyecto o el no poder realizar ciertas funcionalidades del proyecto.

3.2 Análisis de riesgos

- Riesgos tecnológicos**:**

Riesgo	**Probabilidad**	**Efectos**
Limitaciones en el laboratorio	Media	Tolerables
Uso de diferentes versiones por parte de los miembros del grupo	Baja	Tolerables

- Riesgos de personal:

Riesgo	**Probabilidad**	**Efectos**
Personal no cualificado	Alta	Serios
Falta de entendimiento de los miembros del grupo	Baja	Serios
Ausencia de algún miembro del grupo por viaje o enfermedad	Media	Tolerables
Abandono del proyecto por parte de algún miembro del grupo	Baja	Tolerables
Falta de ánimo de trabajo del grupo	Baja	Serios
Incompatibilidad de horarios	Alta	Serios

- Riesgos de organización:

Riesgo	**Probabilidad**	**Efectos**
Cambio de la estructura del grupo	Baja	Tolerables
Cambio del tutor del proyecto	Baja	Serios

Mala coordinación de tareas	Baja	Serios
Pérdida de documentos o archivos	Baja	Serios
Que no se respete la organización jerárquica de los componentes del grupo	Media	Serios
Desconocimiento por parte de algunos de los miembros del grupo de fechas de reuniones	Baja	Serios

- Riesgos de requisitos:

Riesgo	**Probabilidad**	**Efectos**
Inestabilidad de los requisitos	Alta	Serios
Cambio en los requisitos del producto antes de la entrega	Media	Serios
Incumplimiento de requisitos fijados por falta de tiempo o desconocimiento de realización	Baja	Serios

- Riesgos de estimación:

Riesgo	**Probabilidad**	**Efectos**
No saber la fecha de entrega del proyecto con suficiente antelación	Baja	Tolerables
Conflicto de recursos temporales	Baja	Tolerables

3.3 Planificación de riesgos

- Riesgos tecnológicos:

Riesgo	**Estrategia**
Limitaciones en el laboratorio	- Control de recursos empleados

Uso de diferentes versiones por parte de los miembros del grupo	- Ponernos de acuerdo en las versiones que usaremos

- Riesgos de personal:

Riesgo	**Estrategia**
Personal no cualificado	- Investigación y documentación previa de cada tecnología o tema desconocido.
Falta de entendimiento de los miembros del grupo	- Evitar la incompatibilidad de horarios y poner disponibles todas las vías de comunicación posibles con cada miembro - Incentivar las buenas relaciones entre los miembros. Realizar reuniones periódicas para solventar posibles problemas en las relaciones laborales.
Ausencia de algún miembro del grupo por viaje o enfermedad	- Cada uno debe planificarse para tener disponible su parte en el momento que se necesite.
Abandono del proyecto por parte de algún miembro del grupo	
Falta de ánimo de trabajo del grupo	- Motivación del grupo
Incompatibilidad de horarios	- Planificar distribuyendo la carga de trabajo teniendo en cuenta los horarios de los miembros del grupo.

- Riesgos de organización:

Riesgo	**Estrategia**
Cambio de la estructura del grupo	- Si es por falta de personal habrá que organizar las tareas de nuevo entre los restantes componentes del grupo.
Cambio del tutor del proyecto	- Intentar sacar el máximo partido al trabajo realizado hasta ese momento.
Mala coordinación de tareas	- Revisión de los problemas que hubiera

	en la coordinación actual realizando una replanificación para intentar solventarlos.
Pérdida de documentos o archivos	- Rehacer los documentos. Para evitar esto también nos creamos una cuenta en gmail a la cual se iban subiendo las distintas versiones realizadas por cada uno de los miembros del grupo.
Que no se respete la organización jerárquica de los componentes del grupo	- Intentar preguntar a los jefes de cada uno de los departamentos en los que se divide el grupo, en lugar de hablar directamente con otros miembros.
Desconocimiento por parte de algunos de los miembros del grupo de fechas de reuniones	- Dejar mensajes en la comunidad cada vez que se fije la fecha de una reunión y comprometernos todos a revisar los mensajes enviados a la misma frecuentemente.

- Riesgos de requisitos:

Riesgo	**Estrategia**
Inestabilidad de los requisitos	- Revisión de la planificación para intentar solucionar los nuevos problemas que pueden surgir en el desarrollo del proyecto.
Incumplimiento de requisitos fijados por falta de tiempo o desconocimiento de realización	- Revisar con detenimiento los requisitos y, si es necesario, cambiarlos sacrificando las tareas menos prioritarias.
Cambios continuos en las especificaciones	- Partir de una especificación muy clara que se consiga gracias a reuniones entre todos los miembros del grupo para que todo quede claro.

- Riesgos de estimación:

Riesgo	**Estrategia**
No saber la fecha de entrega del proyecto con suficiente antelación	- Ponernos una fecha prevista de finalización antes de lo que se piensa que será la fecha oficial para en todo caso tener terminado el proyecto antes de tiempo, pero en ningún caso fuera de plazo.
Conflictos de recursos temporales	- Compromiso por parte de todos los miembros con el grupo para respetar plazos y en caso de aparecer el riesgo colaborar lo máximo posible en solventarlo para que el retraso sea leve.

3.4 Riesgos encontrados durante el desarrollo del proyecto

- Riesgos de personal
  - Durante el periodo de las vacaciones de Navidad y Semana Santa no estaban todos los miembros del grupo en Madrid para poder hacer reuniones para dividir el trabajo y ver cuales eran los próximos pasos a tomar.
  - Los tres componentes del grupo han realizado durante el curso prácticas de formación en empresas o becas de colaboración con departamentos de la universidad, lo que ha limitado el tiempo disponible para el proyecto.
  - El tiempo que nos han quitado los exámenes y realización de prácticas y trabajos de las diferentes asignaturas en las que estamos matriculados.

- Riesgos tecnológicos:
  - El desconocimiento de XML y de las herramientas anteriores como Xerces y FLTK y su correspondiente instalación en las diferentes plataformas utilizadas.

- El fallo de la conexión a internet en nuestros hogares en algún momento determinado ha ocasionado no poder mandar versiones al resto de integrantes del grupo.
- Problemas con nuestros ordenadores (en algún momento no han arrancado o funcionado correctamente).
- También nos encontramos con el desconocimiento por completo del lenguaje C++ por parte de uno de los miembros del grupo.

- Riesgos de organización:
  - Al principio no sabíamos cómo dividir el trabajo entre los tres miembros del grupo.
  - Falta de una planificación detallada desde el principio, debido en gran manera al desconocimiento del proyecto y del tiempo que nos iba a llevar completar cada tarea propuesta.

3.5 Soluciones tomadas ante a los riegos aparecidos

- Riesgos de personal

  Estos riesgos se han solucionado trabajando en las horas libres que hemos ido teniendo a lo largo del curso (fines de semana, noches, etc).

- Riesgos tecnológicos:

  El desconocimiento de las tecnologías se ha solventado estudiándolas en profundidad (algunas desde el verano) y ayudándonos de tutoriales, libros, etc. Cuando fallaban los ordenadores o la conexión a Internet nos íbamos a trabajar a los laboratorios de la facultad, hasta que el problema se resolvía.

- Riesgos de organización:
  - Según iba avanzando el proyecto íbamos haciendo planificaciones a corto plazo, más realistas y adecuadas al esfuerzo que requería la siguiente tarea a realizar

4. Planificación y Desarrollo

Al principio hicimos una planificación en la que dividimos todo el tiempo del que disponíamos en iteraciones de aproximadamente 4 semanas, planificación altamente optimista debido al desconocimiento del alcance del proyecto. Las iteraciones que pensamos a día 20 de octubre de 2005 son las siguientes:

- 1ª y 2ª iteración: Instalación de la distribución de Linux (Debian) y estudio de algunas librerías como FLTK, GTK+, etc. Tutorial C++, diseño gráfico de la aplicación, UML, compilar y ejecutar ejemplos más complejos.
- 3ª iteración: Estructura general de la aplicación, desarrollo de interfaces y métodos de generación de código.
- 4ª y 5ª iteración: Implementación.
- 6ª iteración: Implementación y pruebas.
- 7ª iteración: Memoria, presentación y pruebas.

A continuación se muestra en una tabla la planificación inicial:

Octubre	1	1	2	2
Noviembre	2	2	3	3
Diciembre	3	3	**V**	**V**
Enero	**V**	4	4	4
Febrero	4	**E**	**E**	**E**
Marzo	5	5	5	5
Abril	6	**V**	6	6
Mayo	6	7	7	7
Junio	7	**E**	**E**	**E**
Julio				

En las casillas numeradas, el número se corresponde con la iteración, las marcadas con una V se corresponden con un periodo de vacaciones y las que tienen una E con un periodo de exámenes.

El desarrollo real ha estado basado en las diferentes entregas que hemos tenido con la tutora del proyecto.

- 1ª iteración (octubre de 2005): Instalación de la distribución de Linux (Debian) y estudio de algunas librerías como FLTK, GTK+, etc.
- 2ª iteración (noviembre 2005): Creación de una primera versión del editor gráfico y añadir funcionalidades como abrir, guardar como, etc. Estudiar XML y ver el funcionamiento de Xerces. División del proyecto en 3 módulos.
- 3ª Iteración (diciembre 2005): Primer prototipo del árbol gráfico del editor donde se van a mostrar los elementos añadidos al proyecto. Primer prototipo también del generador de ficheros XML y del programa que, usando Xerces, recorría estos ficheros XML.
- 4ª Iteración (enero): Aumento de la funcionalidad del editor y por tanto del generador de ficheros XML y del generador de código. En el editor ya existe la opción de crear un proyecto estándar
- 5ª Iteración (17/03/2006): Se añaden todos los botones y los valuators. Comienzo del estudio de los callbacks y de su mejor forma de incluirlos en el código generado.
- 6ª Iteración (27/04/2006): Añadimos la opción de crear callbacks y eventos a los diferentes elemento de la interfaz. También se añaden los bloques de código, los bloques de declaraciones, las clases y las funciones. También se generan dos ficheros XML para cada clase. Se permite hacer resize y seleccionar elementos de la ventana.
- 7ª Iteración (17/05/2006): Primera versión de la memoria del proyecto. Los ficheros XML son un poco más estéticos, ya que los elementos aparecen tabulados y permiten una más fácil comprensión. Se consigue que los módulos 1 y 2 funcionen en Linux. Se cambia la estructura de los callbacks y de los eventos. Primera versión de la aplicación en Linux.
- 8ª Iteración (02/07/2006): Versión final tanto de código como de memoria. Se integran los tres módulos y se reestructura la aplicación. Se añaden estilos a los elementos gráficos que se pueden añadir a un proyecto. Versión estable en Linux.

A la hora de la implementación del código hemos dividido el proyecto en tres partes prácticamente independientes, pero ligadas entre sí por ciertos elementos que ahora comentaremos.

Módulo1: La primera parte es el editor gráfico, que es el módulo encargado de permitir al usuario crear su interfaz gráfica y guardar toda la información de la interfaz en una estructura de datos creada por nosotros a para dicho propósito. De este módulo se ha encargado Alberto del Barrio García.

Módulo 2: El segundo módulo es el enlace entre los otros dos módulos, ya que se encarga de traducir el contenido de la estructura de datos del primer módulo en documentos XML de forma que el tercer módulo pueda trabajar con estos documentos. De este módulo se ha encargado Julio Sevillano Camarero.

Módulo 3: El tercer y último módulo de nuestro proyecto es el encargado de procesar los documentos XML creados por el módulo anterior y generar el código C++ asociado a estos XML que contienen toda la información relativa a la interfaz gráfica desarrollada por el usuario. De este módulo se ha encargado Francisco Javier Rincón Vallejos.

El módulo en el que se ha invertido más tiempo, es el primero, ya que es en el que hemos tenido que definir las estructuras de datos, los diferentes tipos de elementos, la interfaz gráfica, etc.

Los otros módulos se han basado en los elementos definidos en el primero, por lo que no han sido tan difíciles de implementar como el anterior, pero sí que han tenido la dificultad añadida de que trabajan con la tecnología XML, de la cual teníamos total desconocimiento y nos ha supuesto ciertos problemas.

El proceso de gestión de configuración que hemos seguido a la hora de gestionar las diferentes versiones tanto del código como de la memoria es el siguiente:

- Cuando empezamos el proyecto nos creamos una cuenta de correo en GMAIL donde subíamos el trabajo realizado cada vez que teníamos una versión estable.

- Estas versiones también se mandaban a las diferentes cuentas de correo de los miembros del grupo.
- Cada miembro del grupo se encargaba de guardar las versiones de su módulo en su disco duro y de hacer copias de seguridad en CD cada mes.

5. Casos de uso

5.1 Sección Archivo

5.1.1 Nuevo

Permite al usuario crear un nuevo proyecto. En el caso de que haya un proyecto abierto, ofrece la posibilidad de salvarlo.

Después de esto, aparece la ventana de nuevos proyectos. Se ofrece la posibilidad de crear un proyecto vacío, o bien un proyecto estándar, compuesto por una clase que extiende a Fl_Window, un cuadro de texto y un botón.

Se permite seleccionar el directorio del proyecto, así como el nombre. Hasta que el nombre introducido no sea correcto (no vacío y sin espacios en blanco), no se podrá crear un proyecto.

Podemos abandonar esta ventana bien pulsando OK (o tecleando Enter), para lo cual deben cumplirse los requisitos anteriores, en cuyo caso se creará un proyecto, o bien pulsando CANCEL o la X, en cuyo caso no se creará nada. Esta acción de crear un nuevo proyecto se puede realizar también pulsando ctrl +n.

5.1.2 Abrir

Esta opción permite abrir un proyecto anteriormente creado con la aplicación. Al igual que en el caso anterior, se ofrece la posibilidad de salvar el proyecto activo. Esta acción también tiene la posibilidad de realizarse pulsando ctrl+a.

5.1.3 Guardar

Utilizamos esta opción para salvar el proyecto actual, en el directorio de trabajo actual. El proceso consiste en traducir la información, guardada en la estructura de datos del programa, a archivos con formato XML. Se puede realizar también pulsando ctrl +g.

5.1.4 Guardar como

Permite al usuario guardar el proyecto en el directorio de trabajo que seleccione. El proceso es análogo al de *Guardar*. Se puede realizar también pulsando shift + ctrl +x.

5.1.5 Escribir código

Esta función permitirá crear el código C++ asociado a nuestro proyecto, a través de los ficheros XML anteriormente guardados. Se puede realizar también pulsando ctrl + e.

5.1.6 Salir

Utilizamos esta opción para cerrar la aplicación. Se ofrece la posibilidad de guardar el proyecto actual. Se puede realizar también pulsando ctrl + q.

5.2 Sección Editar

5.2.1 Borrar

Esta opción permite al usuario eliminar los items seleccionados. Nótese que si el item tiene hijos, serán eliminados sus hijos también. Se puede realizar también pulsando ctrl + x.

5.2.2 Seleccionar todo

Esta opción permite al usuario seleccionar todos los ítems que se encuentran en el browser. Se puede realizar también pulsando shift + s.

5.3 Sección Ver

5.3.1 Ítems

Con esta opción visualizamos una ventana que contiene botones con cada uno de los ítems que se puede insertar. Cada botón dispone de ayuda contextual, para facilitar la tarea al usuario. La funcionalidad es la misma que accediendo vía menú.

5.3.2 Propiedades del proyecto

Despliega una ventana con las propiedades del proyecto activo, como son el nombre del proyecto y la ruta.

5.4 Sección Añadir

Cada ítem se añadirá a un ítem seleccionado, salvo en el caso de las clases, en las que es preciso que se añadan a la raíz. Así mismo se ofrece la posibilidad de que los ítems sean públicos, protegidos o privados (sólo en el caso de atributos y funciones).

5.4.1 Código

5.4.1.1 Clase

Esta opción permite crear una clase dentro del proyecto. Para ello debe introducirse el nombre de la clase y de la superclase. Si éste segundo está vacío, se asume que no se hereda de ninguna superclase. Si la superclase es Fl_Window, la clase que añadimos es una ventana, y podremos añadirle cualquier elemento que se pueda añadir a Fl_Window.

Restricciones: Los nombres han de ser correctos, y el ítem padre ha de ser el raíz (ninguno seleccionado).

5.4.1.2 Funciones

Esta opción permite añadir funciones al proyecto actual. Para ello debe introducirse un nombre y opcionalmente argumentos y el tipo de retorno. En el caso de que el usuario no introduzca ningún tipo de retorno, se asume que éste es void.

Restricciones: Los nombres han de ser correctos, y el ítem padre ha de ser una clase.

5.4.1.3 Declaraciones

Esta opción permite añadir una declaración al proyecto actual.

Restricciones: Una declaración puede agregarse a una clase, una función, un bloque de declaraciones o un bloque de código. Se deja al usuario la responsabilidad de la corrección de la declaración.

5.4.1.4 Código

Esta opción permite añadir líneas de código al proyecto actual. Para ello hay que introducir el código dentro del cuadro de texto que nos aparecerá.

Restricciones: Sólo podemos añadir código a una función o a un bloque de código. Se deja al usuario la responsabilidad de la corrección del código.

5.4.1.5 Bloque de declaraciones

Esta opción permite añadir un bloque de declaraciones al proyecto actual. Para ello hay que indicar las condiciones de comienzo y finalización del bloque.

Restricciones: Un bloque de declaraciones sólo puede añadirse a una clase, a una función, a un bloque de código o a otro bloque de declaraciones. Se deja al usuario la responsabilidad de la corrección de las condiciones de inicio y fin del bloque de declaración.

5.4.1.6 Bloque de código

Esta opción permite añadir un bloque de código al proyecto actual. Para ello hay que indicar las condiciones de comienzo y finalización del bloque.

Restricciones: Un bloque de código sólo puede añadirse a una función o a otro bloque de código. Se deja al usuario la responsabilidad de la corrección de las condiciones de inicio y fin del bloque de código.

5.4.2 Contenedores

5.4.2.1 Group

Esta opción permite añadir un widget de tipo Fl_Group al proyecto actual. Para ello hay que dar un nombre forzosamente, e introducir otros parámetros si se desea. También se ofrece la posibilidad de asociar código a los eventos de release (con los dos botones del ratón).

Restricciones: Un Group sólo puede añadirse a otro contenedor. Si se añade a un contenedor tipo Tabs, las dimensiones quedan fijadas a las del Tabs, no pudiendo el usuario modificarlas. El nombre y las dimensiones deben ser correctos.

NOTA: entendemos por contenedor una clase que extiende a Fl_Window, un Group o un Tabs.

5.4.2.2 Tabs

Esta opción permite añadir un widget de tipo Fl_Tabs al proyecto actual. Para ello hay que dar un nombre forzosamente, e introducir otros parámetros si se desea. También se ofrece la posibilidad de asociar código a los eventos de release (con los dos botones del ratón).

La función de este tipo de contenedores es a su vez contener a varios Groups, cada uno de los cuales forma una pestaña del Tabs.

Restricciones: Un Tabs sólo puede añadirse a otro contenedor. El nombre y las dimensiones deben ser correctos.

5.4.3 Botones

En esta sección podemos agregar un botón al proyecto actual. Para ello hay que dar un nombre forzosamente, e introducir otros parámetros si se desea. Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un botón sólo puede añadirse a una clase que extienda a Fl_Window y a un Group. El nombre y las dimensiones deben ser correctos.

Tipos de botones: Button, Retturn_Button, Light_Button, Check_Button, Repeat_Button, Round_Button.

5.4.4 Valuators

En esta sección podemos agregar un objeto de tipo Valuator al proyecto actual. Para ello hay que dar un nombre obligatoriamente, e introducir otros parámetros si se desea. Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un Valuator sólo puede añadirse a una clase que extienda a Fl_Window y a un Group. El nombre y las dimensiones deben ser correctos.

Tipos de Valuators: Slider, Scrollbar, Value_Slider, Adjuster, Counter, Dial, Roller, Value_Input, Value_Output.

5.4.5 Texto

En esta sección podemos agregar un objeto de tipo cuadro de texto al proyecto actual. Para ello hay que dar un nombre obligatoriamente, e introducir otros parámetros si se desea. Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un cuadro de texto sólo puede agregarse a una clase que extienda a Fl_Window y a un Group. El nombre y las dimensiones deben ser correctos.

Dentro de esta categoría, entrarían los siguientes cuadros de texto: File_Input, Multiline_Input, Multiline_Output (estas dos últimas opciones permiten introducir más de una línea de texto), Text_Display, Text_Editor.

5.4.6 Menu

5.4.6.1 Menu_Bar

Esta opción permite al usuario agregar una barra de menú al proyecto actual. Para ello hay que dar un nombre obligatoriamente, e introducir otros parámetros si se desea. Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un Menu_Bar sólo puede agregarse a una clase que extienda a Fl_Window y a un Group. El nombre y las dimensiones deben ser correctos.

5.4.6.2 Menu_Button

Esta opción permite al usuario agregar un Menu_Button al proyecto actual. Para ello hay que dar un nombre obligatoriamente, e introducir otros parámetros si se desea. Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un Menu_Button sólo puede agregarse a una clase que extienda a Fl_Window y a un Group. El nombre y las dimensiones deben ser correctos.

5.4.6.3 Choice

Esta opción permite al usuario agregar al proyecto actual un cuadro para seleccionar determinados ítems, que en un principio se visualiza como una pestaña minimizada. Para ello hay que dar un nombre obligatoriamente, e introducir otros parámetros si se desea. Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un Choice sólo puede agregarse a una clase que extienda a Fl_Window y a un Group. El nombre y las dimensiones deben ser correctos.

5.4.6.4 Submenu

Esta opción permite al usuario agregar un submenú a un menú del proyecto actual. Para ello hay que dar un nombre

obligatoriamente, e introducir otros parámetros si se desea. Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un Submenu sólo puede agregarse a un objeto de tipo menú. El nombre y las dimensiones deben ser correctos.

5.4.6.5 MenuItem

Esta opción permite al usuario agregar un ítem a un menú del proyecto actual. Para ello hay que dar un nombre obligatoriamente, e introducir otros parámetros si se desea. Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un MenuItem sólo puede agregarse a un objeto de tipo menú. El nombre y las dimensiones deben ser correctos.

5.4.7 Explorador

Esta sección permite al usuario agregar al proyecto un cuadro selector de ciertas opciones. Este cuadro dispone de un scroll que aparece si la disposición de las opciones supera las dimensiones de éste. Para agregarlo hay que dar un nombre obligatoriamente, e introducir otros parámetros si se desea. Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un objeto de tipo Explorador sólo puede agregarse a una clase que extienda a Fl_Window y a un Group. El nombre y las dimensiones deben ser correctos.

Tipos de explorador: Fl_Browser, Fl_Check_Browser, Fl_File_Browser.

5.4.8 Otros

En esta sección se permite añadir otro tipo de objetos, que no pertenecen a las categorías anteriores.

5.4.8.1 Box

Esta opción sirve para agregar una caja, que podría contener una etiqueta. Para ello hay que dar un nombre obligatoriamente, y otros parámetros si se desea. Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un Box sólo puede agregarse a una clase que extienda a Fl_Window y a un Group. El nombre y las dimensiones deben ser correctos.

5.4.8.2 Clock

Esta opción permite agregar un reloj al proyecto actual. Para ello hay que dar un nombre obligatoriamente, e introducir otros parámetros si se desea. Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un Clock sólo puede agregarse a una clase que extienda a Fl_Window y a un Group. El nombre y las dimensiones deben ser correctos.

5.4.8.3 Help_View

Esta opción permite agregar un cuadro de ayuda al proyecto actual. Para ello hay que dar un nombre obligatoriamente, e introducir otros parámetros si se desea. Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un Help_View sólo puede agregarse a una clase que extienda a Fl_Window y a un Group. El nombre y las dimensiones deben ser correctos.

5.4.8.4 Progress

Esta opción permite agregar una barra de progreso al proyecto actual. Para ello hay que dar un nombre obligatoriamente, e introducir otros parámetros si se desea.

Asimismo, se ofrece la posibilidad de asociar código para los eventos de release (con los dos botones del ratón).

Restricciones: Un Progress sólo puede agregarse a una clase que extienda a Fl_Window y a un Group. El nombre y las dimensiones deben ser correctos.

5.4.9 Consideraciones adicionales

- Definimos nombre correcto como aquel que no contiene espacios.
- Los nombres de los ítems no pueden estar repetidos dentro de una misma clase.
- Los nombres de las clases de un mismo proyecto tienen que ser distintos entre sí y distintos del nombre del proyecto.
- El resto de parámetros que se permiten introducir para cada widget está dividido en tres grupos:
  - Pestaña general. Aquí se incluyen las características más comunes de los widgets: nombre, dimensiones, etiqueta, ayuda contextual y visibilidad (público, protegido o privado). Los campos de dimensiones permanecen inhabilitados en el caso de añadir Fl_Groups a Fl_Tabs.
  - Pestaña callbacks. Código que se debe ejecutar en caso de que se produzca un evento activado en cada tipo de widget, y activación de los eventos. Para las ventanas se permiten los dos eventos de release (soltar ratón tanto con botón izquierdo como con derecho), close (cerrar ventana) y focus (la ventana toma el foco o control del programa). Para el resto de widgets sólo se permiten los eventos de release, excepto para los Menu_Items y para los Submenus, que solo tienen el release con el botón izquierdo.
  - Pestaña estilo. Aquí se incluyen todas las características visuales de los widgets: fuente, tamaño, color, alineación y estilo de la etiqueta, estilo y color de la caja que rodea al

widget cuando está presionado (como en un botón) y cuando no, y fuente, tamaño y color del texto.

- En el caso de la pestaña estilo, hay algunos campos que se inhabilitan según los casos, a saber:
  - Los campos fuente, tamaño y color del texto sólo se habilitan para los widgets de tipo campo de texto.
  - Los campos de estilo y color de down-box, o caja cuando se presiona, sólo se habilitan para los botones o los menús.
- Las dimensiones de los widgets se entienden como correctas cuando las coordenadas son mayores o iguales que cero y se ajustan a la resolución de la pantalla. En el caso de las ventanas, simplemente se comprueba que la anchura y la altura de las mismas sea mayor que cero.

5.5 Ayuda

5.5.1 Acerca de

Se muestra una ventana con una pequeña información sobre los integrantes del grupo y la directora del proyecto.

5.5.2 Sobre

Se ofrece una ayuda en formato html, que contiene aspectos básicos de la aplicación y permite instruir al usuario mediante la realización de un ejemplo sencillo. Esta ayuda es relativa a nuestra aplicación, pero también contiene un enlace a la ayuda de FLTK.

5.5.3 Manual

Se ofrece un enlace al manual de la librería FLTK, para que el usuario pueda familiarizarse con todas las posibilidades que permite dicha librería.

5.6 Acciones con el ratón

Aparte del menú de la aplicación, el ratón constituye una parte muy importante de la misma. Con él podemos seleccionar, arrastrar y redimensionar visualmente los widgets, siempre que las dimensiones sean correctas. Del mismo modo, haciendo doble clic sobre un widget (o un ítem del browser) tenemos acceso a todas sus propiedades, mientras que con un clic simple solamente lo seleccionamos.

Además, cada vez que el ratón pasa por un componente de la aplicación, muestra un mensaje de ayuda contextual de gran utilidad para el usuario.

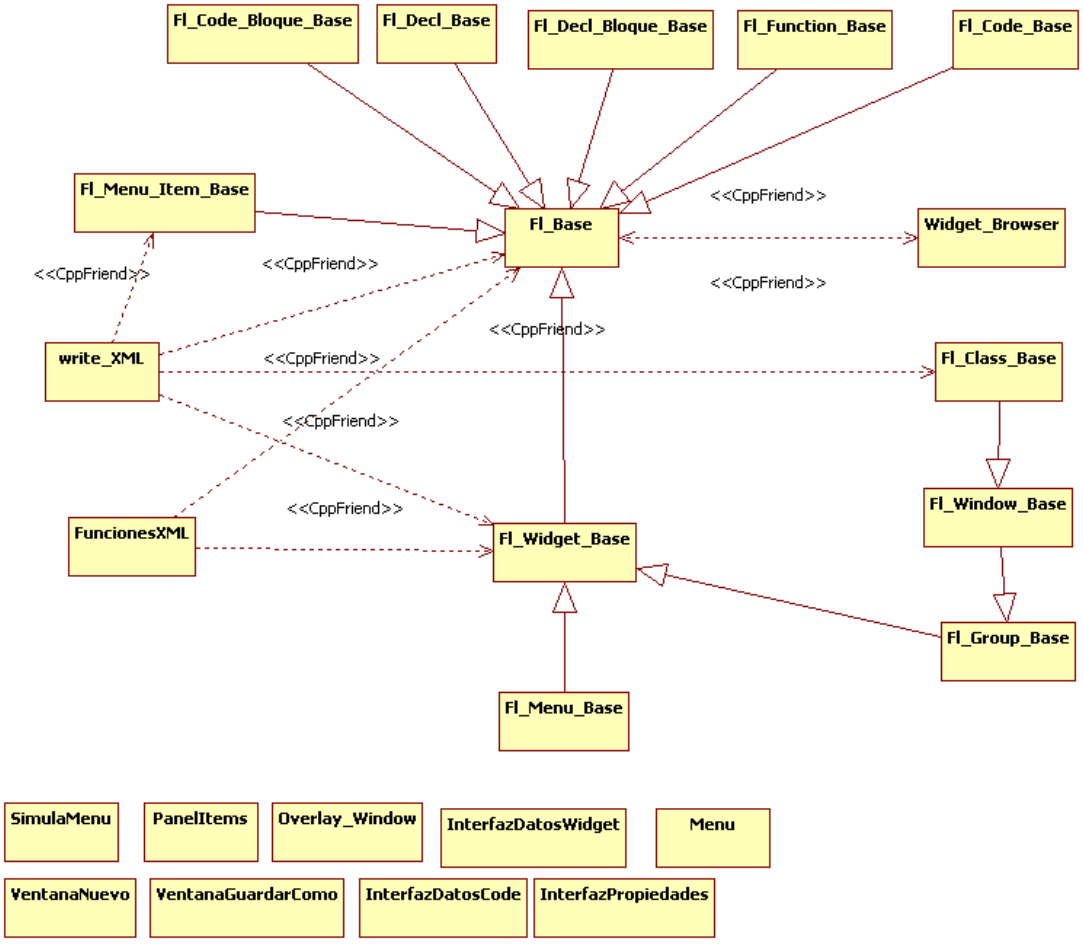
6. UML

6.1 Diagramas Estructurales de Clases

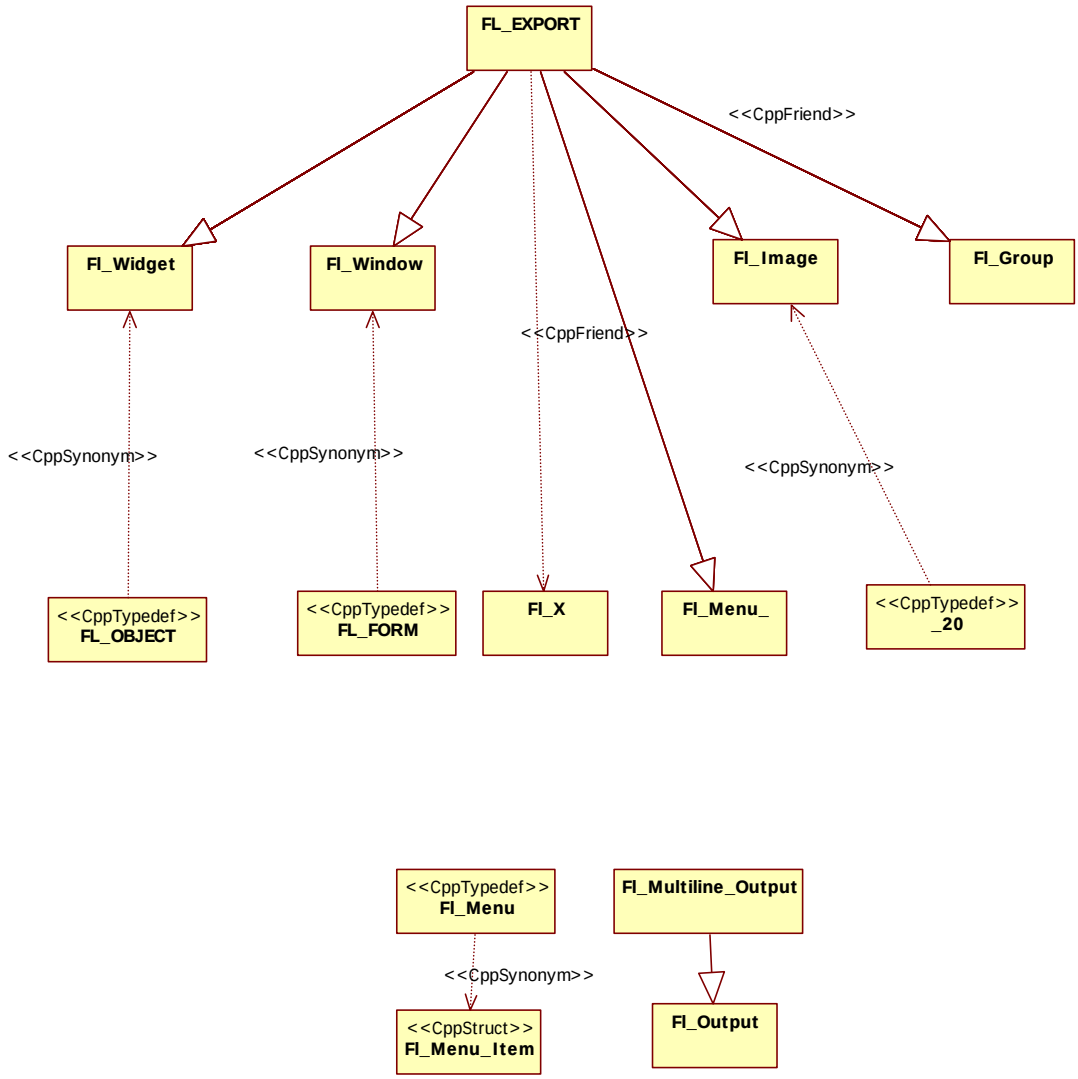
Leyenda:

- La flecha discontinua indica que la clase desde la que sale la flecha tiene como friend class la clase a la que llega la flecha (Sobre estas flechas viene en un cuadro de texto "<<Cppfriend>>").
- La flecha continua va desde un hijo hasta su clase padre.

A continuación el diagrama de clases de la aplicación en el que se muestra las relaciones de herencia de las clases.



Ahora mostramos un pequeño diagrama de clases de la librería FLTK para ver como está organizada.

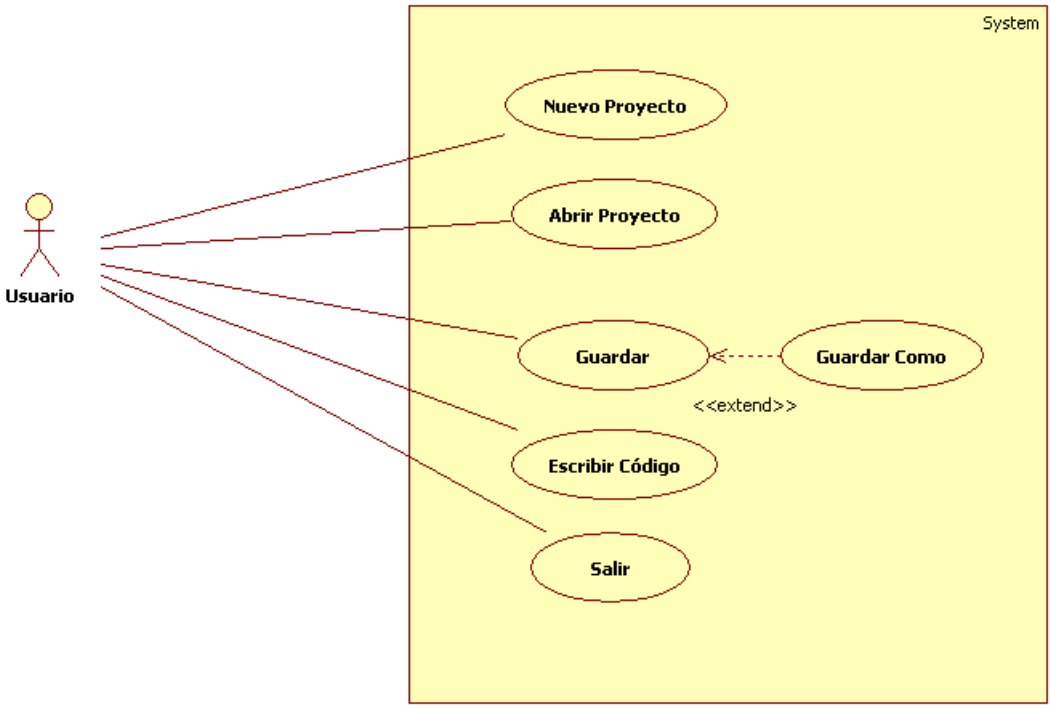


6.2 Diagramas de Comportamiento

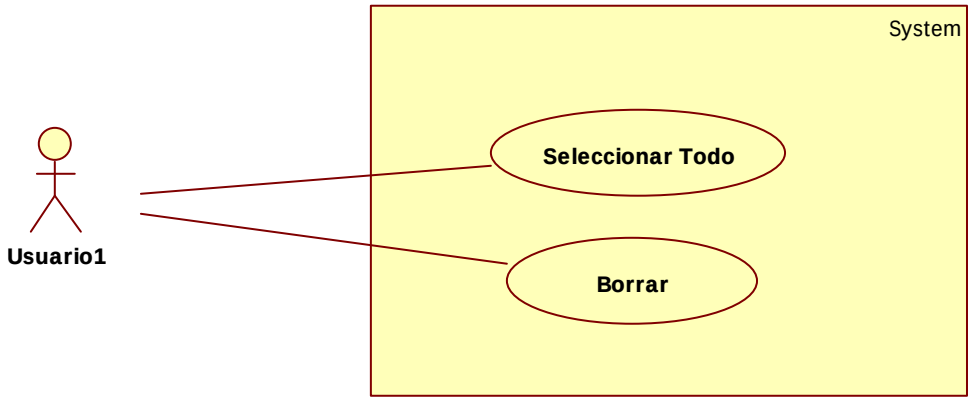
6.2.1 Diagrama de Casos de Uso

En el diagrama de casos de uso aparece un único actor, que es el usuario de la aplicación y los correspondientes casos de uso que describen cómo se comporta la aplicación ante las acciones que puede realizar el usuario. Los casos de uso los hemos agrupado en cinco secciones, que son las cinco pestañas en las cuales se pueden realizar acciones de nuestro editor.

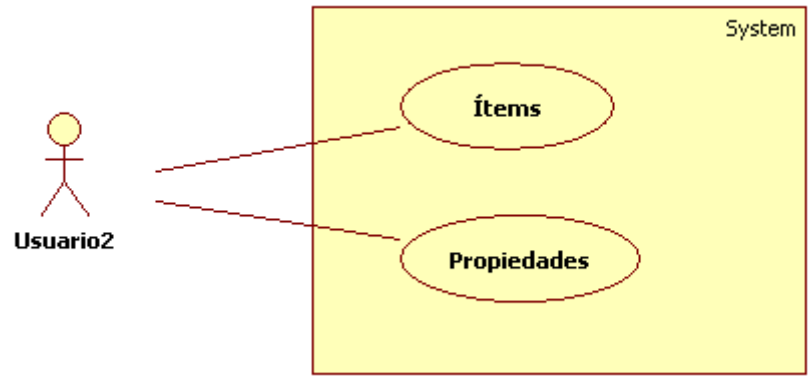
- Casos de Uso de la sección Archivo:



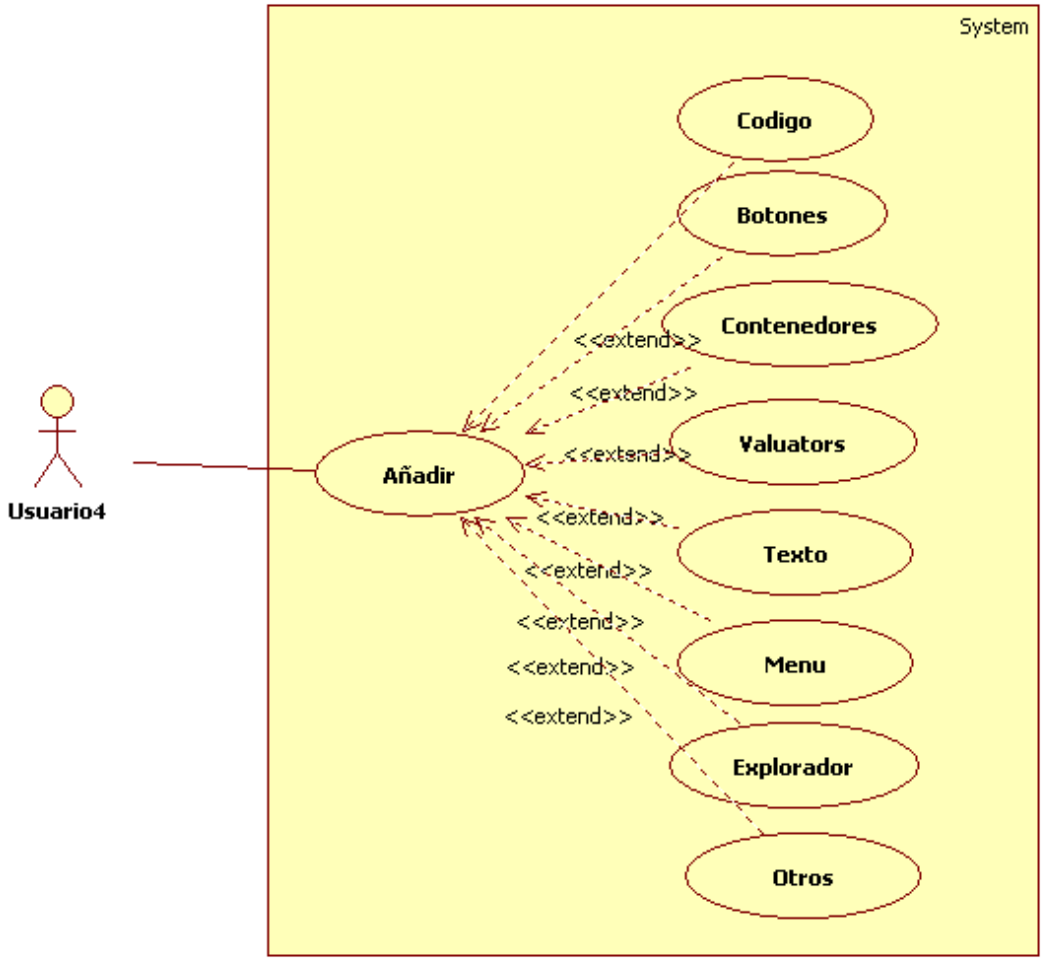
- Casos de Uso de la sección Editar:



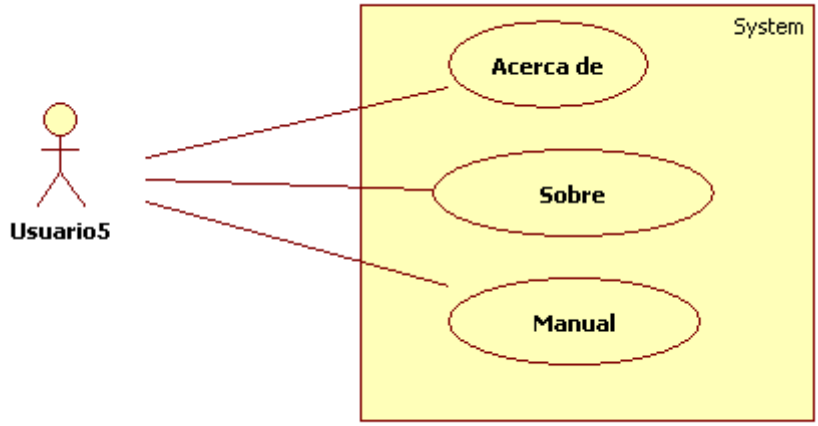
- Casos de Uso de la sección Ver:



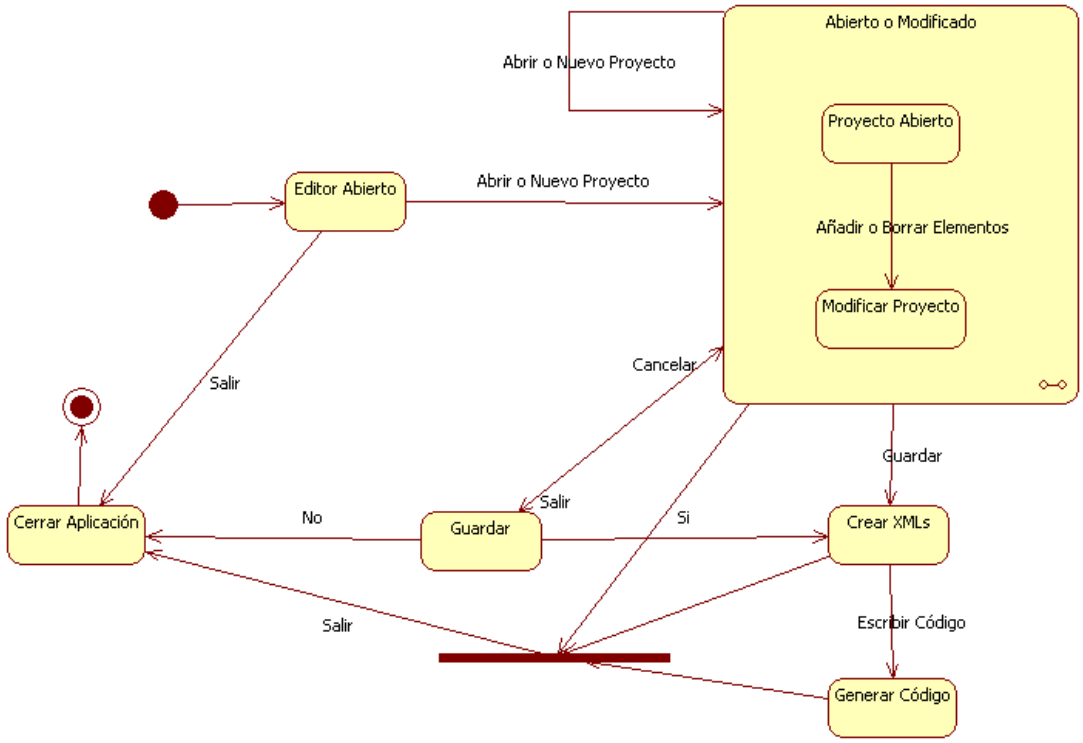
- Casos de Uso de la sección Añadir:



- Casos de Uso de la sección Ayuda:



6.2.2 Diagrama de Estados



Este diagrama muestra las diferentes fases por las que puede pasar nuestro proyecto. Una vez abierto el editor, o creamos un fichero nuevo o abrimos uno guardado anteriormente. En este estado podemos hacer lo que queramos con nuestro proyecto, añadir elementos, cambiarlos de sitio, etc. Si volvemos a abrir un proyecto o a crear uno nuevo volvemos a este estado.

Si pulsamos guardar se crearán los ficheros XML ya explicados anteriormente, pasando al estado “Crear XMLs”. Si pulsamos salir pasamos a un estado “Guardar” en el que se nos ofrece antes de salir de la aplicación guardar el proyecto con el que hemos estado trabajando para no perder ese trabajo. Si pulsamos “Si” vamos al estado “Crear XMLs”, si pulsamos “No” salimos de la aplicación y si pulsamos “Cancelar” volvemos al estado anterior. Estando en el estado “Crear XMLs” podemos generar el código C++ pasando al estado “Generar Código”. Desde los estados “Abierto o Modificado”, “Crear XMLs” o “Generar Código” si pulsamos salir vamos a “Cerrar Aplicación” que es el estado final de nuestra aplicación.

7. Implementación

Como hemos descrito en el apartado de Planificación y Desarrollo tenemos tres módulos prácticamente independientes, pero ligados entre sí por ciertos elementos. A continuación los vemos en detalle.

Para ilustrar mejor la funcionalidad de cada módulo, vamos a diseñar una calculadora y en cada módulo explicaremos los pasos que se dan hasta generar el código C++.

7.1 Módulo 1: Interfaz Gráfico

El editor gráfico se compone esencialmente de dos partes: la interfaz propiamente dicha, y la estructura de datos que recoge la información del proyecto.

- Interfaz gráfica

La interfaz gráfica está formada por una ventana principal que contiene un menú, desde el cual acceder a las diversas funcionalidades que ofrece la aplicación, y por una estructura gráfica de árbol que denominaremos *browser*, en el cual mostraremos la lista de elementos del proyecto.

El menú ofrece las opciones de crear proyecto, abrir, guardar, propiedades del proyecto, etc como cualquier editor, y opciones específicas de nuestra aplicación, como son añadir los distintos elementos a nuestro proyecto. Además, permite visualizar un panel con estos elementos, para facilitar la tarea al usuario. Podemos ver todas las acciones explicadas con mayor profundidad en el apartado de Casos de uso.

Por otro lado, también disponemos de una serie de mecanismos para que el usuario pueda interactuar gráficamente con la aplicación, como son la selección, el arrastre y el redimensionamiento de los *widgets* o elementos gráficos.

Cabe destacar que el algoritmo de selección empleado es una versión del Cohen-Sutherland. Permitimos selección simple y selección múltiple aunque, a la hora de añadir elementos al proyecto, solamente se puede hacer si hay un único ítem

seleccionado. Respecto al redimensionamiento y arrastre de widgets, hemos de tener en cuenta las restricciones en los valores de las dimensiones explicadas en los casos de uso.

Además de estas tres acciones, el editor permite seleccionar gráficamente al pulsar sobre un widget con un clic simple. Si el clic es doble, se mostrará una ventana con las propiedades del widget. Esto es válido tanto al actuar sobre el browser como sobre los widgets creados por el usuario.

En el caso de los Menu_Item, hemos de destacar que el clic simple sobre el menú que lo contiene, provoca el lanzamiento del "Simulador de Menús", una ventana del programa que deja al usuario visualizar como quedaría el menú que está construyendo, permitiéndole desplegar todos los Submenu y Menu_Item. Esta opción sólo es válida al interactuar sobre los widgets y no sobre el browser.

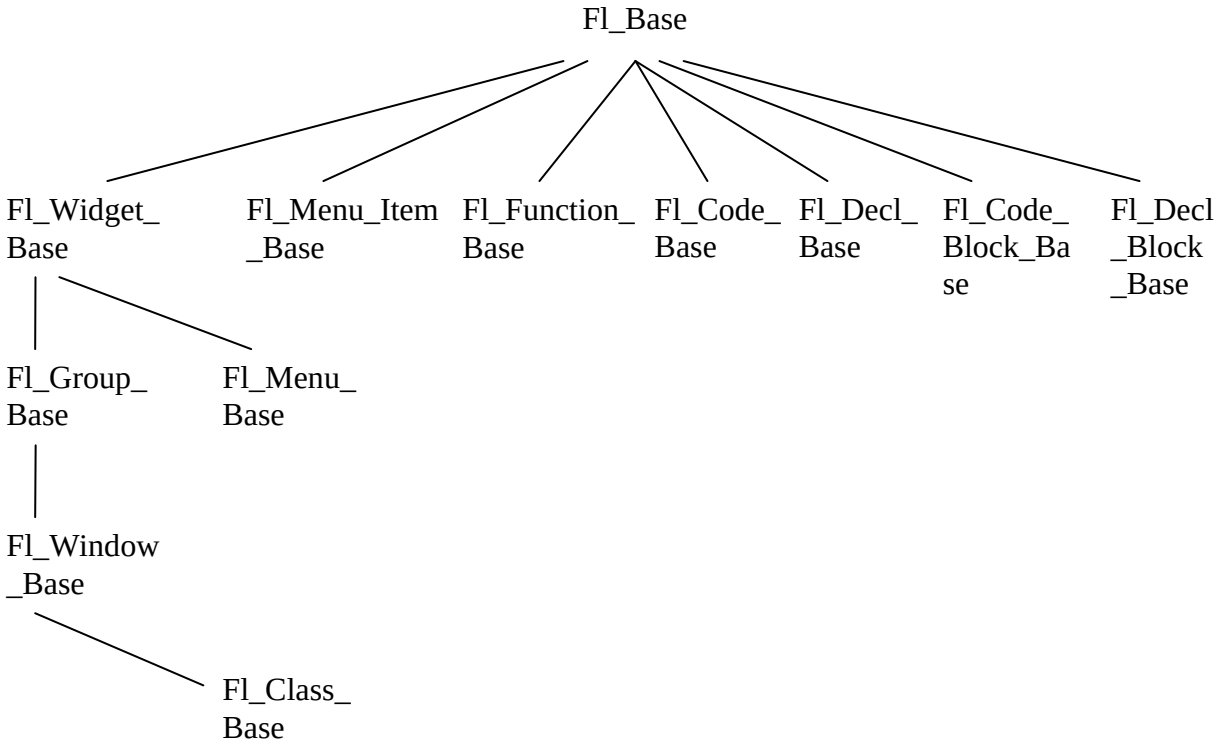
Por otro lado, la interfaz dispone de un sistema de ayuda muy completo. Primeramente unos tutoriales tanto de la aplicación en sí como de la librería fltk, los cuales incluyen el diseño de ejemplos sencillos y, en segundo lugar, una ayuda basada en mensajes de error (en el caso de las restricciones) y en *tooltips* o ayuda contextual.

- Estructura de datos

Para guardar toda la información relativa a los ítems de que consta el proyecto, usamos una estructura de árbol simulada por una lista doblemente enlazada, con punteros estáticos al primer y al último elemento, así como un puntero (también estático) al ítem activo en cada caso, y un puntero al nodo padre.

Disponemos de un conjunto de clases nodo, que constituyen esta estructura. Dependiendo del tipo de elemento que queramos agregar, usaremos uno u otro tipo de nodo para guardar la información específica de cada ítem.

Para facilitar el trabajo y utilizar un paradigma de programación como el orientado a objetos, introducimos una jerarquía de clases entre estos nodos.



Tal y como podemos observar, partimos de un nodo padre denominado Fl_Base. Este tipo de nodo contiene toda la información básica que debe tener cualquier elemento de nuestra lista, como nombre, tipo, etc, así como los punteros anteriormente mencionados. Un detalle importante es que el nombre será el identificador de la variable una vez que se genere el código.

De esta clase heredan Fl_Function_Base, Fl_Code_Base, Fl_Code_Block_Base, Fl_Decl_Base y Fl_Decl_Block_Base que, implementan nodos que se corresponden con funciones, código, bloques de código, declaraciones y bloques de declaraciones, respectivamente, es decir, todo lo relativo a estructuras de código que pueden utilizarse en la programación.

De Fl_Base también hereda directamente Fl_Menu_Item_Base, que es un nodo que contiene un Fl_Menu_Item, el cual es una estructura que utiliza FLTK para guardar la información relativa a los ítems de los menús. Utilizamos este nodo para los tipos Fl_Menu_Item, y Fl_Submenu (que no es un tipo estándar de FLTK, sino un Fl_Menu_Item con un flag de submenú activado).

Por otro lado, de Fl_Base también hereda Fl_Widget_Base. La principal característica de este tipo de nodo es que contiene un puntero a un *widget* (ventanas, botones, cuadros de texto, barras de menú, etc), por lo que usaremos este tipo de nodo para agregar los elementos gráficos.

A su vez, de Fl_Widget_Base hereda Fl_Group_Base que es una clase que utilizamos para los elementos denominados contenedores. En este grupo se encuentran Fl_Group y Fl_Tabs.

De Fl_Group_Base hereda Fl_Window_Base, que es un nodo creado únicamente para las ventanas.

Fl_Class_Base es el nodo correspondiente a las clases. La herencia de Fl_Window_Base se debe a que nuestro objetivo es utilizar una ó ninguna ventanas por clase, de tal forma que entenderemos como si fueran ventanas a las clases que hereden de Fl_Window, y como clases normales al resto.

Las clases que hereden de Fl_Window (valor que el usuario puede introducir), serán denominadas clases gráficas, y en ellas podemos insertar todo tipo de widgets y estructuras de código.

Las clases que no hereden de Fl_Window, serán consideradas como clases no gráficas a las que, por tanto, sólo podremos añadir estructuras de código.

Un punto interesante de esta jerarquía es que es fácilmente ampliable. Deja la puerta abierta a añadir nuevos tipos de nodo sin mayor complicación.

A continuación vamos a mostrar los atributos de cada tipo de nodo, siguiendo la jerarquía:

Todos los Fl_Base tienen los siguientes atributos:

```
char* name;//Nombre del Fl_Base
```

```
char* label;//Etiqueta
char* callback;//Callback
char* user_data_type;//Tipo de datos que contiene el Fl_Base
Fl_Base *parent; //Padre del nodo
bool new_selected; //Indica si el item está seleccionado
bool open;      //Indica si el nodo está abierto ó cerrado, para la visualización en
                //el browser
bool visible; //Indica si el nodo es visible ó no
bool public_; //Indica si un Fl_Base es público ó no. Por defecto a true
int level; //Indica la profundidad del nodo en el árbol
Fl_Base *next, *prev;//Nodos siguiente y previo
static Fl_Base *first, *last; //Nodos primero y último
static Fl_Base *current; //Fl_Base sobre el que se puede actuar. Nodo actual
int numHijos;//Número de hijos
```

Los Fl_Widget_Base además incluyen los siguientes atributos:

```
Fl_Widget *widget;//Objeto que contendrá toda la información de un widget
int box,down_box;//Tipo de caja, con presión y sin presión, del widget
int align_index;//Índice del fl_choice de la interfaz del widget
int label_type_index;//Índice del fl_choice de la interfaz del widget.
```

Los Fl_Group_Base no contienen información adicional con respecto a los Fl_Widget_Base, pero forman una clase porque contienen a todos los widgets que son contenedores y, en futuras ampliaciones podría ser conveniente introducir atributos específicos a dicho tipo de elemento.

Los Fl_Window_Base sólo presentan nuevos atributos relativos a la visualización de los widgets contenidos en la ventana del Fl_Window_Base, para poder realizar las acciones de selección, arrastre ó redimensionamiento de los mismos, a saber:

```
int xs1,ys1;//Coordenadas del vértice superior izquierdo del rectángulo de
selección ó de resize
```

int xs2,ys2;//Coordenadas del vértice inferior derecho del rectángulo de selección ó de resize

bool puls,drag,drop,resize,doble_click;//Flags booleanos

bool norte,sur,este,oeste;//Flags para las direcciones de movimiento

int dx, dy;//Desplazamiento del widget con respecto al ratón en el momento de moverlo

int mdx, mdy;//Posición del ratón dentro de la ventana

Fl_Widget* widget_used;//Widget arrastrado ó que hará resize ó seleccionado

int lado;//Longitud del lado del cuadrado del rectángulo de selección

int hgap,vgap;//Margen de error para los lados de los widgets

Los Fl_Class_Base sólo añaden dos atributos (con respecto a Fl_Window_Base) para guardar el nombre de la clase padre:

char* parent_class;//Clase padre

bool isWindow;//Flag para distinguir si la clase hereda de Fl_Window ó no.

Los Fl_Function_Base añaden 2 atributos al Fl_Base:

char* args;//Argumentos de la función

char* return_type;//Tipo de retorno

En los Fl_Decl_Base usamos el atributo name de Fl_Base, para guardar la declaración. Por tanto, no usamos atributos adicionales.

En los Fl_Decl_Bloque_Base tenemos dos atributos adicionales:

char* ifCond;//Condición del bloque, lo usamos como name del Fl_Base

char* endIf;//Fin del bloque

En los Fl_Code_Base tenemos un atributo para el código. Si el código es corto, éste será el name (atributo de Fl_Base), si no sólo utilizaremos como name los 25 primeros caracteres, seguidos de " …".

char* codigo;//Código que introduzca el usuario

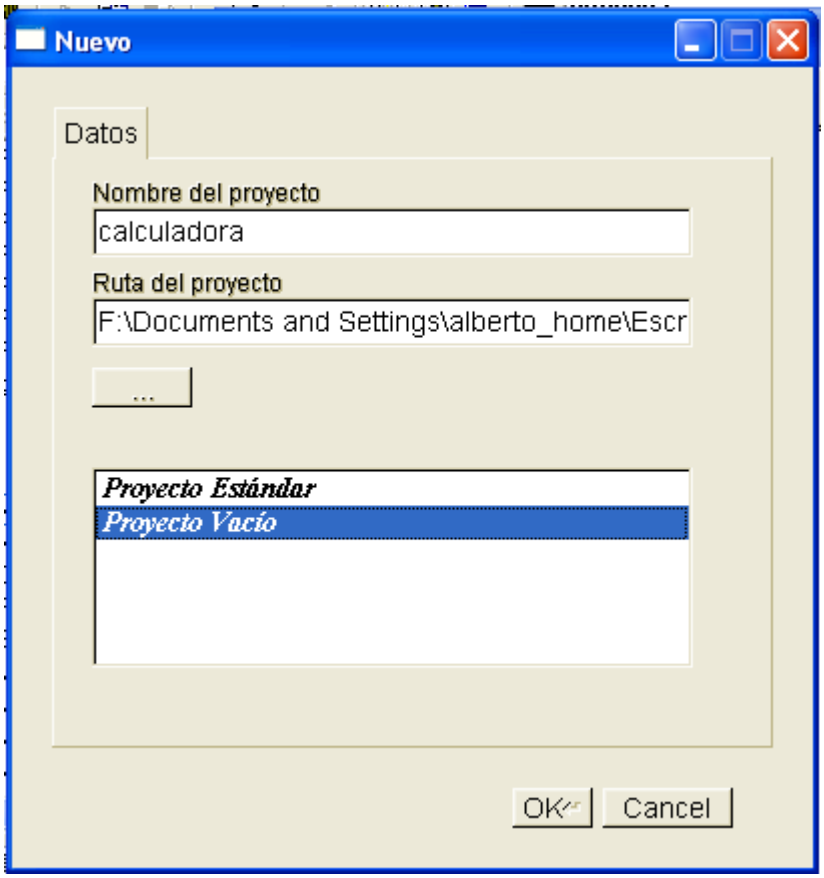
En los Fl_Code_Bloque_Base tenemos también los atributos de inicio y finalización de bloque:

char* ifCond;//Condición del bloque, lo usamos como name del Fl_Base
char* endIf;//Fin del bloque

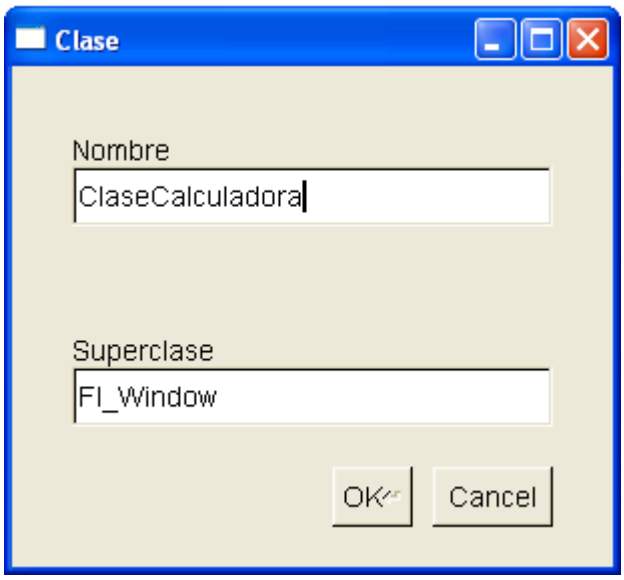
- **Diseño de la calculadora**

Como hemos introducido previamente, vamos a mostrar cómo actuaría el módulo 1 mediante el diseño de una calculadora.

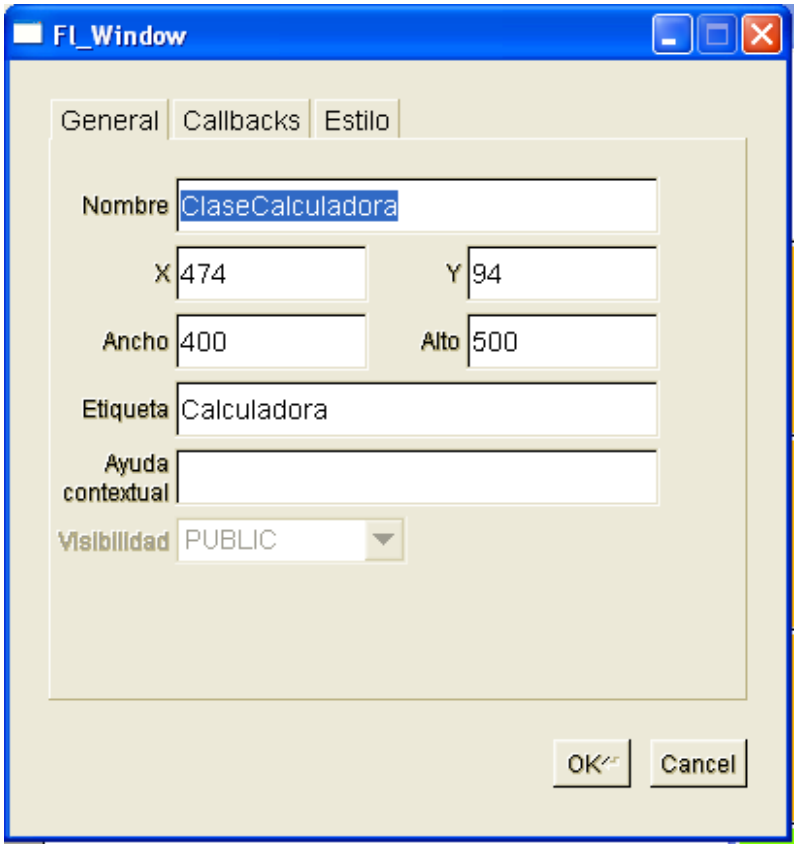
En primer lugar abrimos el editor y creamos un proyecto vacío, con el nombre de calculadora.



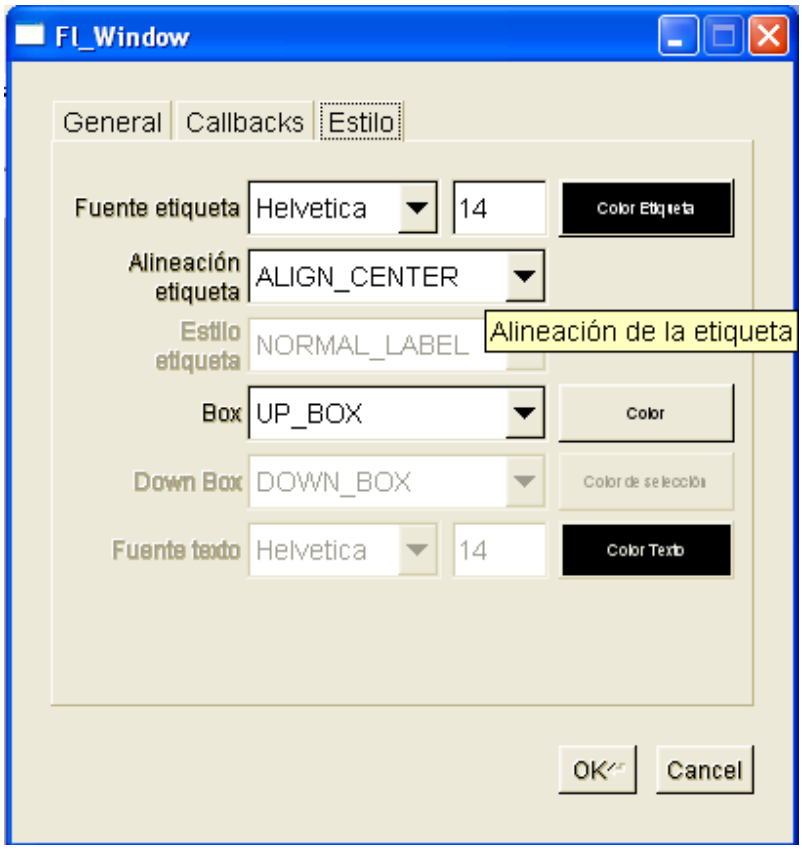
En segundo lugar creamos una clase que extienda de Fl_Window, bien con el panel gráfico o bien con la opción Añadir→ Código → Clase.



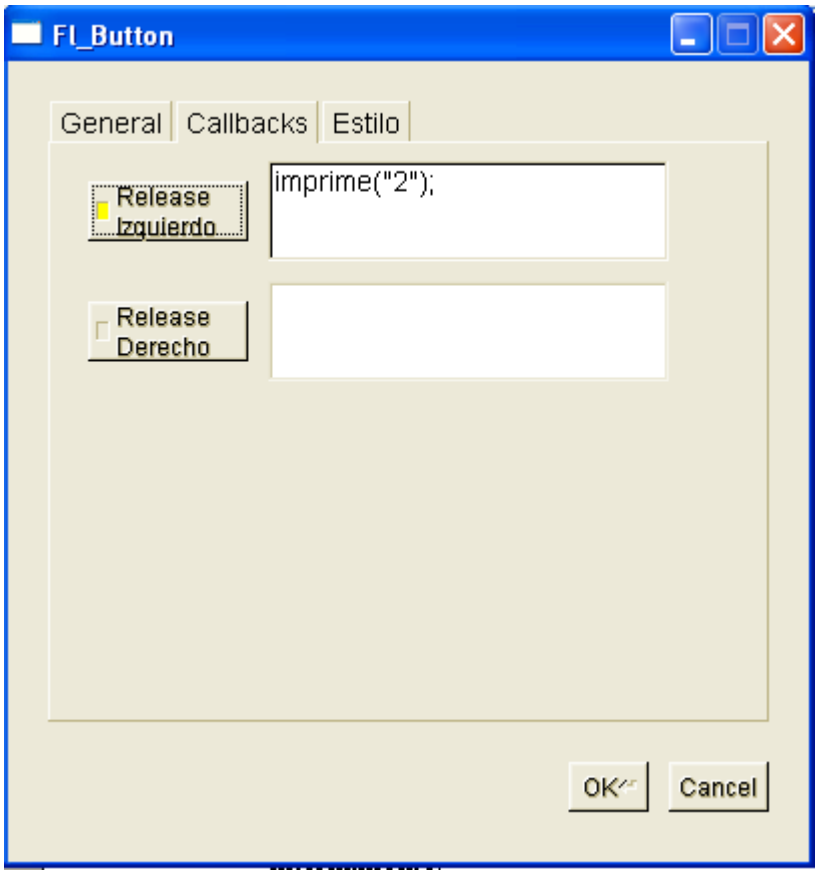
Pulsamos OK, y nos aparecerá la ventana de propiedades. Introducimos nombre, dimensiones, y demás.



Introducimos las opciones de estilo y pulsamos OK.

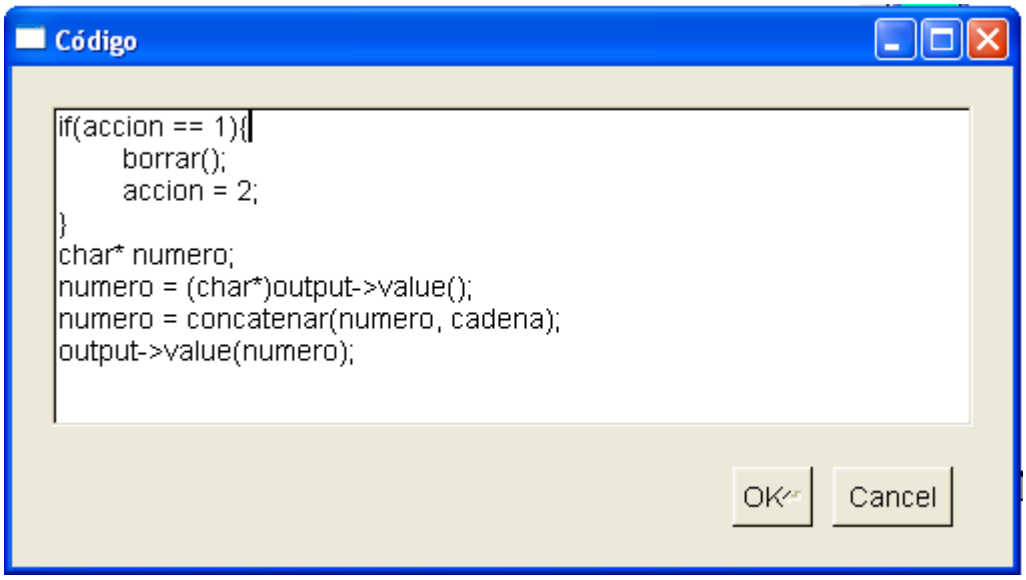


A continuación insertamos los botones y el cuadro de texto de la calculadora, de la misma manera que con la ventana anteriormente. Por ello, ahora nos fijaremos sólo en la pestaña Callback, en la que introducimos la llamada a una función *imprime()*.



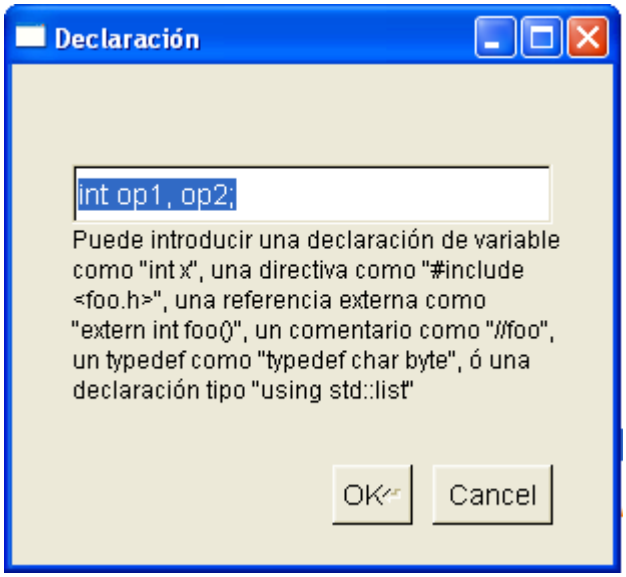
Después nos creamos esta función *imprime()*, con los parámetros adecuados.

E insertamos el código de la misma: (Nótese que output es el nombre del Multicine_Output, por lo que lo utilizamos para llamarlo en el código)

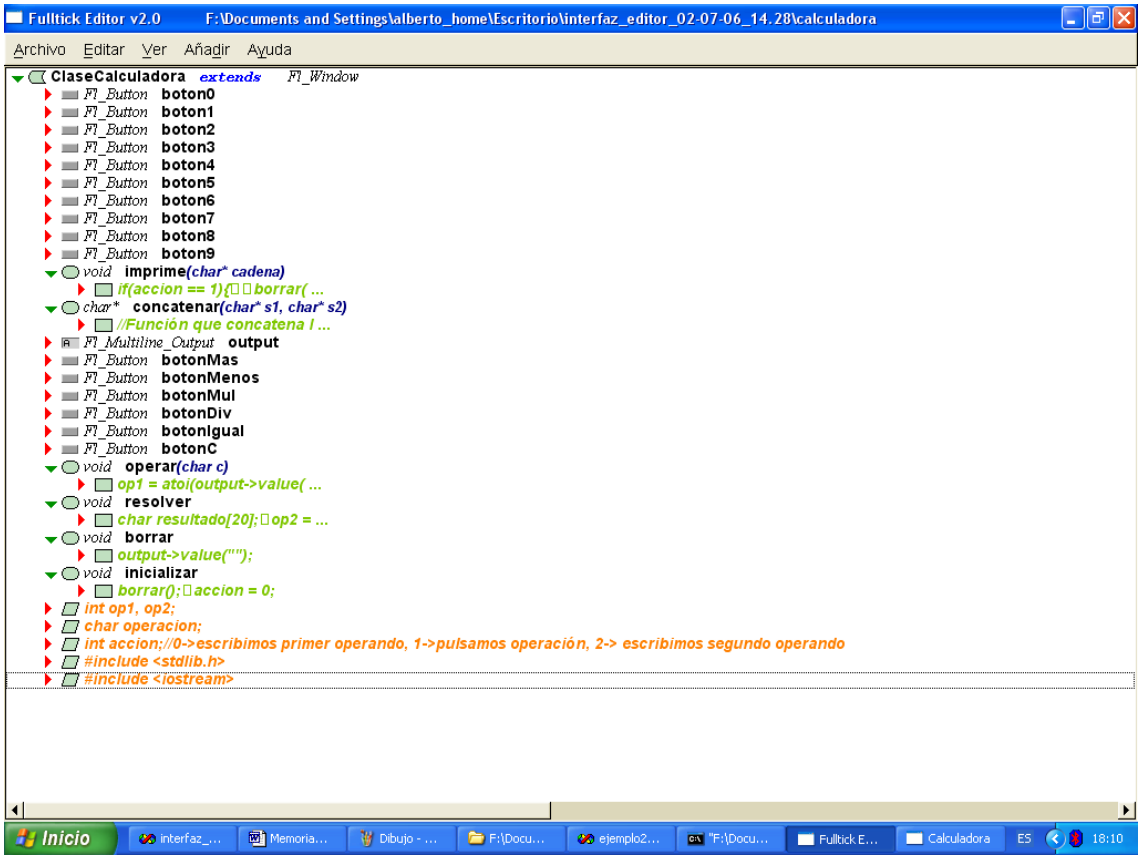


De la misma manera, nos creamos otras funciones específicas con su código asociado, e introduciendo la llamada en la pestaña Callback del botón correspondiente.

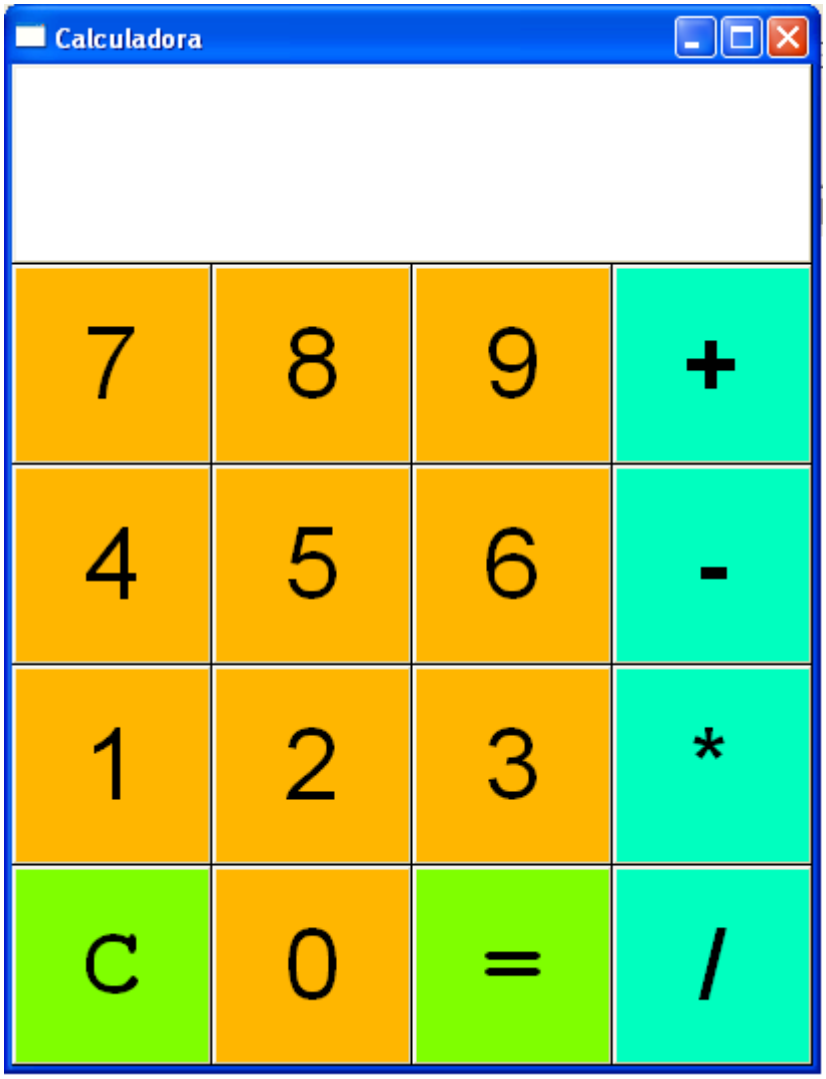
Además de todo esto, necesitamos declarar algunas variables e incluir algunas librerías de c++.



Finalmente, el aspecto que presentaría nuestro browser sería el siguiente:



Y la calculadora resultante quedaría de la siguiente manera:



Hemos visto todo lo relativo a la interfaz gráfica, ahora estudiaremos cómo procesamos internamente estos datos.

Partimos de una lista vacía. En un primer momento hemos creado una clase, por lo que los punteros estáticos quedarían así:

first= Fl_Class_Base(“ClaseCalculadora”);

last= Fl_Class_Base(“ClaseCalculadora”);

current = 0;

Los atributos de este nodo quedarían como sigue:

name = “ClaseCalculadora”;

user_data_type = “Fl_Window”; // Porque la clase extiende de Fl_Window

parent = 0; //La clase no tiene padre, cuelga del raíz

prev = 0; //Nodo anterior

next = 0; //Nodo siguiente

callback = “###”;//La ventana no tiene ningún callback, pero a lo sumo puede tener 4, cada uno de los cuales se separa con el carácter ‘#’.

…

Si pulsamos la ventana para insertar cualquier otro elemento, el puntero estático *current* pasaría a apuntar al nodo_ventana.

current= Fl_Class_Base(“ClaseCalculadora”);

A continuación hemos insertado el botón del 0, que pertenece a los Fl_Widget_Base. Los punteros se actualizarían así:

first= Fl_Class_Base(“ClaseCalculadora”);

last= Fl_Widget_Base(“boton0”);

Los atributos del nuevo nodo quedarían así:

name = “boton0”;

user_data_type = “Fl_Button”; // Tipo del botón

parent = Fl_Class_Base(“ClaseCalculadora”); //La clase es el padre del botón.

prev = Fl_Class_Base(“ClaseCalculadora”); //Nodo anterior

next = 0; //Nodo siguiente

callback = “imprime("0");#”; //Un widget puede tener hasta dos callbacks. En este caso, boton0 sólo tiene evento de release izquierdo.

…

El puntero next del nodo anterior también se actualizaría:

Fl_Class_Base(“ClaseCalculadora”)->next = Fl_Widget_Base(“boton0”);

De la misma manera, hemos insertado los otros 9 botones. Siguiendo el orden de los elementos que tenemos en el browser, hemos insertado la función *imprime()*.

Actualización de punteros estáticos:

first= Fl_Class_Base(“ClaseCalculadora”);

last= Fl_Function(“imprime”);

current = Fl_Class_Base(“ClaseCalculadora”);

Los atributos del nuevo nodo quedarían así:

name = “imprime”;

return_type = “”; //Equivale a void

args = “char* cadena”; //Argumentos de la función

parent = Fl_Class_Base(“ClaseCalculadora”); //La clase es el padre de la función.

prev = Fl_Widget_Base(“boton9”); //Nodo anterior

next = 0; //Nodo siguiente

…

El puntero next del nodo anterior también se actualizaría:

Fl_Widget_Base(“boton9”)->next = Fl_Function_Base(“imprime”);

A continuación hemos insertado el código de la función.

Seleccionamos la función, que será el padre del código. Actualización de punteros estáticos:

first= Fl_Class_Base(“ClaseCalculadora”);

last= Fl_Code_Base(“if(accion == 1)\nborrar(…”);

current = Fl_Function_Base(“imprime”);

Los atributos del nuevo nodo quedarían así:

name = “if(accion == 1)\nborrar(…”;

codigo = “if(accion == 1){

borrar();

accion = 2;

}

char* numero;

numero = (char*)output->value();

numero = concatenar(numero, cadena);

output->value(numero);”

parent = Fl_Function_Base(“imprime”); //La función es el padre del código.

prev = Fl_Function_Base(“imprime”); //Nodo anterior

next = 0; //Nodo siguiente

…

El puntero next del nodo anterior también se actualizaría:

Fl_Function_Base(“imprime”)->next = Fl_Code_Base(“if(accion == 1)\nborrar(…”);

De la misma manera insertaríamos el resto de ítems. Pero veamos qué pasaría si eliminásemos alguno de los mismos.

Supongamos que terminamos de insertar nuestros ítems. Por ejemplo, vamos a eliminar el boton8. Lo seleccionamos y efectuamos la operación Editar → Borrar, o bien pulsamos ctrl + x.

Actualización de punteros estáticos:

first= Fl_Class_Base("ClaseCalculadora");

last= Fl_Decl_Base("#include <iostream>");

current = 0;//El puntero actual pasaría a apuntar a null.

El puntero next del nodo anterior al que eliminamos y el puntero prev del siguiente también se actualizarían:

Fl_Widget_Base("boton7")->next = Fl_Widget_Base("boton9");

Fl_Widget_Base("boton9")->prev = Fl_Widget_Base("boton7");

Procedemos de la misma manera, seleccionamos el último item, que es una declaración, y lo borramos.

Actualización de punteros estáticos:

first= Fl_Class_Base("ClaseCalculadora");

last= Fl_Decl_Base("#include <stdlib.h>");

current = 0;//El puntero actual pasaría a apuntar a null.

El puntero next del nodo anterior al que eliminamos también se actualizaría:

Fl_Decl_Base("#include <stdlib.h>")->next = 0;

Ahora, supongamos que queremos borrar una de las funciones, por ejemplo, *imprime()*. La seleccionamos y aplicamos borrar. Veremos que se borra la función, y el código de la misma, ya que es hijo de la función.

Actualización de punteros estáticos:

first= Fl_Class_Base("ClaseCalculadora");

last= Fl_Decl_Base("#include <stdlib.h>");

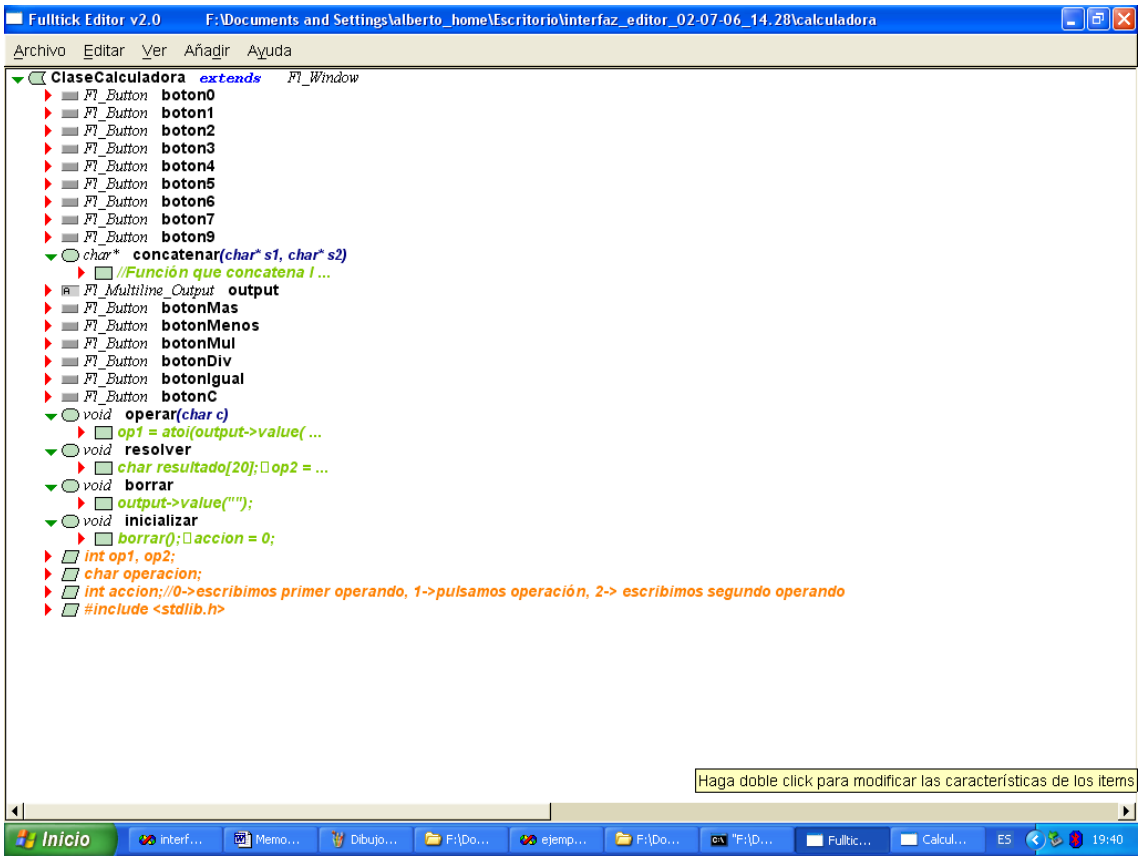
current = 0;//El puntero actual pasaría a apuntar a null.

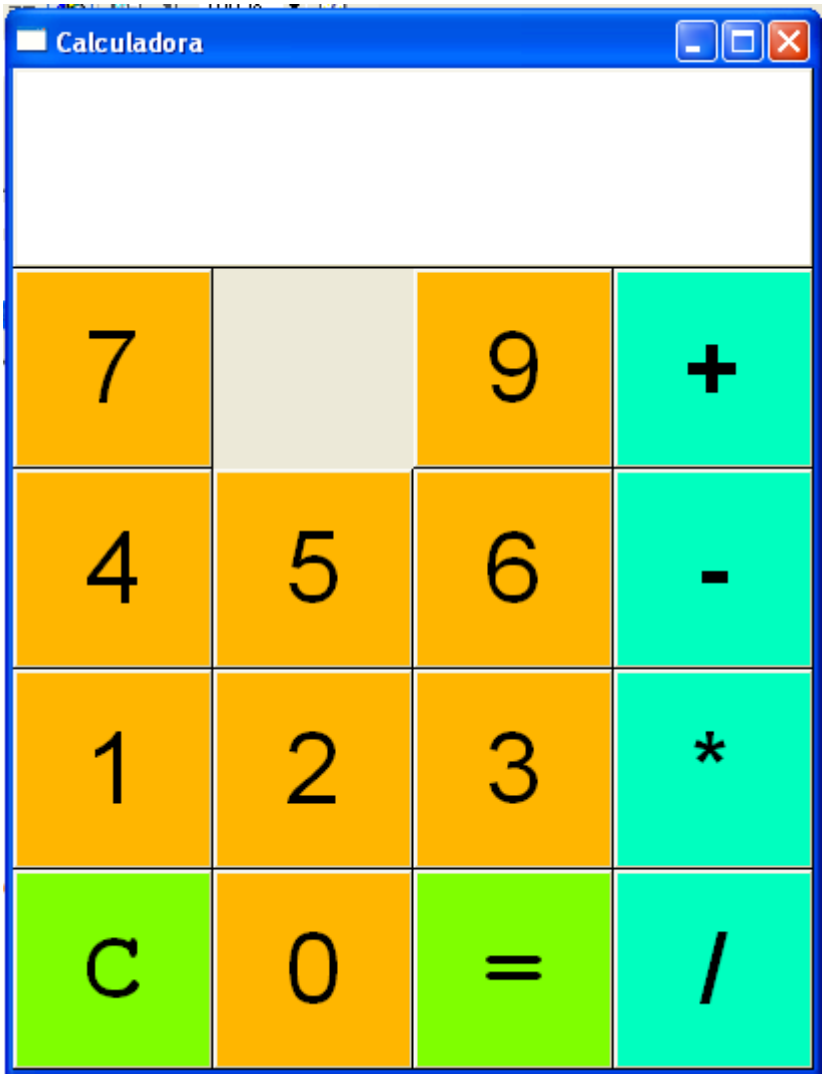
El puntero next del nodo anterior al que eliminamos y el puntero prev del nodo posterior al último hijo que eliminamos también se actualizarían:

Fl_Widget_Base("boton9")->next = Fl_Function_Base("concatenar");

Fl_Function_Base("concatenar")->prev = Fl_Widget_Base("boton9");

Veamos cómo quedarían tanto el browser como la calculadora, después de eliminar estos ítems.





7.2 Módulo 2: Generador de XML

Este módulo empieza a trabajar cuando el usuario haya creado su interfaz gráfica. Para llamar a este módulo el usuario debe hacer "Archivo → Guardar" o "Archivo → Guardar Como". Ahora explicamos como funciona este módulo internamente.

Cuando un usuario crea un proyecto, se generarán los siguientes ficheros:

- Un fichero XML por cada clase que el usuario haya creado.
- Un fichero XML para el proyecto general, en el que habrá una función main, en la que se incluyen todas las clases existentes en el proyecto.
- Un fichero de texto en el que se guarda el nombre de los XML generados (Este fichero de texto tiene extensión .fl).

Al pulsar el usuario “Guardar” (con “Guardar Como” sería igual solo que mostrando un explorador para que el usuario elija donde guardar el proyecto) se llama a la función “guardarFich()” de “Menu.cpp”. En esta función nos creamos un objeto de tipo write_XML y se llama a la función generaXML(), que a partir del primer elemento de la estructura de datos guardada en el módulo 1, procesa esa estructura y genera los ficheros XML correspondientes.

Para esto nos servimos de la función procesa_lista(), que lo primero que hace es crear la carpeta donde se va a guardar el proyecto.

Por ejemplo, supongamos que el usuario quiere guardar su proyecto en “C:\Proyectos\NuevoProyecto”. Si esta carpeta existe, crea dentro de ello una carpeta “\XML” y es dentro de ésta donde guarda los XML. Si la carpeta no existiera, la crea y también crea la carpeta “\XML”. Entonces, los ficheros que crea este módulo se guardarían en “C:\Proyectos\NuevoProyecto\XML\”.

Comentar que los objetos write_XML tienen entre otros atributos, tres char* que son “includecadena”, “cabecera” y “clasexml”. Estos atributos nos sirven para ir guardando en ellos el código que vamos a escribir en los ficheros XML para cada clase.

También tenemos los atributos ficheroProyecto e includeProyecto, que es donde iremos escribiendo el código que tendrá el fichero XML del proyecto.

Después de crear la carpeta, en “procesa_lista( )” vamos recorriendo todos los elementos que se encuentran en nuestra estructura de datos, y dependiendo del tipo de elemento que estemos tratando realizaremos unas acciones u otras. Para ir avanzando en la lista y seguir un orden nos vamos fijando en el atributo “level” de los elementos.

Los diferentes tipos que nos podemos encontrar al procesar un elemento son: clase (con o sin superclase), atributo, grupo, tab, función, declaración, código, bloque de declaraciones y bloque de código.

Entonces lo primero que hacemos en “procesa_lista( )” es comprobar si el elemento es una clase (El primer elemento de nuestros proyectos siempre será una

clase). Si es una clase llamamos a la función “encontrada_nueva_clase_Inicializar( )”. En esta función liberamos el contenido de nuestros tres atributos, crea el nombre de los ficheros XML que vamos a generar, e inicializamos los atributos includecadena y clasexml para el caso de que existan más clases en nuestro proyecto, y también creamos los nombres del fichero XML de esta clase y dónde lo vamos a guardar. También llamamos a la función “escribe_En_Fichero_Proyecto_Widget( )” que se encarga de escribir en el XML de nuestro proyecto que hemos encontrado una nueva clase y escribe la información relativa a ésta.

Una vez procesada la clase, pasamos al siguiente elemento de la lista y llamamos a la función “comprobar_tipo_y_escribir()”. En el atributo de la clase “elementoproceso” se encuentra el elemento actual de la estructura que estamos procesando.

En esta función nos creamos un elemento del tipo “FuncionesXML” que además de comprobar si el elemento es un atributo(por atributo nos referimos a un elemento de la interfaz gráfica como un botón, campo de texto,… pero no bloques de código o declaraciones) también comprueba si en el atributo “includecadena” ya hay un include del tipo de este elemento(si ya hay un atributo de este tipo no queremos que nos vuelva a añadir un include de este tipo) y si no existe lo añade a “includecadena”. En el caso de que el elemento que estemos tratando sea un atributo llamamos a la función “write_code_atributos(Fl_Base *elemento)” que se encarga de guardar en el atributo “clasexml” el código asociado a este atributo.

La estructura que tiene el código XML asociado a un atributo (en este ejemplo el atributo es un botón que tendrá asociado mostrar dos mensajes por consola si el botón es pulsado con el botón izquierdo o derecho del ratón) es la siguiente:

```
<attribute>
  <type>Fl_Button*</type>
  <name>Button_1</name>
  <visibility>public</visibility>
  <parent>MiClase</parent>
  <label>OK</label>
  <position>
        <x>130</x>
        <y>150</y>
  </position>
  <size>
```

```
            <w>40</w>
            <h>25</h>
    </size>
    <label_font>0</label_font>
    <label_size>14</label_size>
    <label_color>56</label_color>
    <label_align>0</label_align>
    <label_align_index>0</label_align_index>
    <label_type>FL_NO_LABEL</label_type>
    <label_type_index>1</label_type_index>
    <box>2</box>
    <box_color>49</box_color>
    <down_box>0</down_box>
    <down_box_color>49</down_box_color>
    <callback>
      <event>FL_RELEASE1</event>
      <call>Button_1_left_cb()</call>
    </callback>
    <callback>
      <event>FL_RELEASE3</event>
     <call>Button_1_right_cb()</call>
    </callback>
  </attribute>
  <function>
    <name>Button_1_left_cb</name>
    <visibility>public</visibility>
    <body>
            <code_block>printf("release izquierdo");</code_block>
    </body>
  </function>
  <function>
    <name>Button_1_right_cb</name>
    <visibility>public</visibility>
    <body>
            <code_block>printf("release derecho");</code_block>
    </body>
  </function>
```

Como se puede observar los atributos van entre las etiquetas <attribute>… </attribute>". Dentro de esto tenemos las siguientes etiquetas:

- <type>: Indica el tipo de atributo que es.
- <name>: Nombre del atributo.
- <visibility>: Indica si el atributo es público o privado.
- <position>: Indica la posición x e y del atributo.
- <size>: Indica el tamaño, el ancho y el alto del atributo.
- <callback>: Contiene la información relativa a los callback, que es lo que se debe hacer ante diferentes eventos del ratón. Recordamos que hemos contemplado como posibles eventos del ratón el "release" con el botón izquierdo

y derecho del ratón, y el "focus" y el "close", aunque estos dos últimos eventos sólo están disponibles para los atributos del tipo "Fl_Window". Para los Menu_Item y Submenu sólo existe el release con el botón izquierdo del ratón.

- <tooltip>: Esta etiqueta sirve para la ayuda contextual, que consiste en un mensaje que nos muestra información sobre este atributo cuando pasemos el ratón por encima de él.
- Otras etiquetas: Además de las etiquetas comentadas anteriormente, existen otras relativas a los estilos, es decir, al tipo y tamaño de las fuentes, los colores, etc. (esto se consigue mediante las funciones obtener_Estilo_Widget y obtener_Estilo_Menu_Item).

La etiqueta "<event>" contiene que tipo de evento desencadena este callback y puede ser uno de los siguientes: fl_release1, fl_release3, fl_focus y fl_close.

La etiqueta "<call>" contiene el nombre de la función que se desencadenará.

Justo después de la etiqueta "</attribute>" si este atributo tiene callbacks, van las funciones asociadas a un callback van dentro de las etiquetas "<function>… </function>" y tienen la siguiente estructura:

- <name>: Nombre de la función.
- <body>: Cuerpo de la función.
- <code_block>: Contiene el código que ha escrito el usuario para que se ejecute cuando se produzca el evento elegido.

La función "write_code_atributos(Fl_Base *elemento)" va escribiendo el código XML asociado a este atributo y en el caso de que tenga un callback llama a las funciones "write_eventos(char* callback,Fl_Base *elemento)" que se encarga de escribir lo que hay entre las etiquetas "<callback>" y también utiliza la función "write_funcion_callback(char* callback,Fl_Base *elemento)" que se encarga de escribir el código de la función asociada a ese callback. El código asociado a los callbacks viene dentro de una cadena de texto que siempre va a tener "#" que sirven de elementos separadores (los elementos separadores de las ventanas son "###", ya que las ventanas permiten 4 posibles eventos, mientras el resto de atributos permiten sólo 2, excepto los Menu_Item y los Submenu que sólo permiten un evento). Entre cada almohadilla irá el

código asociado a cada evento que haya seleccionado el usuario para ese atributo de los anteriormente comentados (fl_release1, fl_release3, fl_focus y fl_close). Para obtener el contenido de esta cadena utilizamos una función auxiliar “split”.

Por convenio a la hora de los eventos, hemos decidido que el usuario escriba en los campos habilitados para tal efecto el código que quiere que se ejecute cuando se produzca ese evento, y la aplicación internamente lo que hace es guardar todo ese código en una función, que será la que se lance cuando se produzca dicho evento.

A la hora de escribir código en el fichero XML, cuando se trata de código libre, es decir, en los elementos donde el usuario puede escribir lo que quiera, usamos una función auxiliar “convertir_cadena_a_formato_xml(char*)”, que examina el código introducido por el usuario, y comprueba si tiene caracteres que son reservados para el XML, formando parte de su sintaxis, y si existen estos caracteres, los sustituye por su correspondiente entidad en XML. Los caracteres que sustituimos son:

ENTIDAD	CARACTER
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

Volviendo a la función ““comprobar_tipo_y_escribir(Fl_Base *elemento)”, en el caso de que no sea un atributo, hacemos una distinción de casos comprobando los diferentes tipos que puede tener el elemento, como puede ser un menú, menú ítem,una función, una clase, etc. Si es una clase salimos de esta función, ya que como hemos comentado, cuando encontramos una clase debemos crear un XML nuevo y añadir esta información en el XML del proyecto general y llamamos a la función “volcar_en_fichero” que lo que hace es escribir en los ficheros XML el contenido de los atributos “cabecera”, “includecadena” y “clasexml”. También salimos de esta función

en el momento en que al pasar al siguiente elemento de la lista para procesar este sea un null, lo que significará que ya hemos recorrido la lista entera.

En el caso de que el tipo del elemento sea una función llamamos a la función "write_code_funcion(Fl_Base *elemento)". La estructura del código XML para las funciones es:

```
<function>
        <name>Funcion_1</name>
        <type>int</type>
        <visibility>public</visibility>
        <param>int x, char* s</param>
        <body>
                … … …. … ….
        </body>
</function>
```

Como se puede observar una etiqueta va entre las etiquetas "<function>… </function>" y dentro tiene etiquetas para el nombre de la función, la visibilidad, el tipo de lo que devuelve la función, los parámetros que se le pasan como argumentos y entre las etiquetas "<body>…</body>" se encuentra el cuerpo de la función.

Al llamar a la función "write_code_funcion(Fl_Base *elemento)" vamos recorriendo todos los elementos que pueda haber dentro de una función, como son declaración, código, bloque de declaraciones y bloque de código. Cuando nos encontramos con un elemento de estos se llama a la función "write_code_TipoElemento(Fl_Base *elemento)" que generarán el código XML que debemos escribir al procesar este elemento.

Si el elemento encontrado es una declaración llamamos a la función "write_code_declaracion". La estructura del código XML asociado a una declaración es:

```
<declaracion>
            <cuerpo_declaracion>int x;</cuerpo_declaracion>
</declaracion>
```

Como se puede ver la declaración va entre las etiquetas "<declaracion>… </declaracion>" y dentro tiene solo una etiqueta para el cuerpo de la declaración.

En el caso que el usuario escriba en una declaración “#include…”, la aplicación interpreta que el usuario quiere hacer un include, por lo que esto no aparecerá como una declaración en el XML, sino que aparecerá en la sección de includes con la siguiente notación:

```
<include>
      <nameincludeuser>&lt;stdio.h&gt;</nameincludeuser>
</include>
```

En este caso el usuario había puesto en una declaración lo siguiente:
#include <stdio.h>

La aplicación ha interpretado que lo debe meter en la sección de includes y además ha pasado los caracteres reservados de XML a su correspondiente entidad, con la función que se ha comentado anteriormente.

Los elementos del tipo “código” generan un código XML con la siguiente estructura:

```
<code_block>//código prueba dentro de bloque de código</code_block>
```

Los elementos del tipo “bloque declaraciones” tienen la siguiente estructura:

```
<dec_block>
            <ifcond>if (1&gt;2) </ifcond>
            <endif>endif</endif>
</dec_block>
```

Los bloques de declaraciones van entre las etiquetas “<dec_block>…</dec_block>” y dentro tiene etiquetas para la visibilidad de la declaración. Entre las etiquetas “</ifcond>…<endif>” iría el código de los posibles hijos que tuviera el bloque.

Para escribir el código XML asociado a los bloques de declaraciones se utiliza la función “write_code_bloque_declaracion(Fl_Base *elemento)”, que se sirve de la función “procesa_hijos_bloque_declaracion(Fl_Base *elemento)” para recorrer los hijos

que pueda haber dentro del bloque, ya que dentro de los bloques pueden existir hijos del tipo "declaración" y "bloque de declaraciones". Cuando acabamos de procesar los hijos de este bloque, pasamos al siguiente elemento de la lista que se encuentre al mismo nivel que el bloque de declaraciones o a un nivel inferior. Para ver cual es el próximo elemento a procesar nos servimos de las funciones auxiliares "siguiente_hermano(Fl_Base* elemento)" y "siguiente_nodo_nivel_inferior(Fl_Base* elemento)". Como dentro de un bloque puede haber varios hijos entre los cuales puede haber más bloques, con estas funciones controlamos cuál es el siguiente elemento que tenemos que procesar de nuestra estructura.

Los elementos del tipo "bloque código" son similares a los bloques de declaraciones, pero pueden tener hijos de más tipos diferentes, como son "declaraciones, código, bloque de declaraciones y bloque de código". La estructura de los XML es como la de los bloques de código:

```
<block>
	<ifcond>if (3>2){</ifcond>
	... ... ...
	<endif>}</endif>
</block>
```

Para escribir el código XML asociado a un bloque de código se utiliza la función "write_code_bloque_codigo(Fl_Base *elemento)", que se sirve de la función "procesa_hijos_bloque(Fl_Base *elemento)" para recorrer los hijos que pueda haber dentro del bloque, que hace lo mismo que la función "procesa_hijos_bloque_declaracion(Fl_Base *elemento)" solo que pueden existir más tipos de elementos dentro un bloque de código que en un bloque de declaraciones. También utiliza las funciones auxiliares "siguiente_hermano (Fl_Base* elemento)" y "siguiente_nodo_nivel_inferior(Fl_Base* elemento)" para obtener el siguiente elemento a procesar. Al igual que en los bloques de declaraciones el código XML de los hijos de un bloque de código va dentro de las etiquetas "</ifcond>…<endif>". Un ejemplo de bloque de código con hijos es:

```
<block>
	<ifcond>if (2=2)</ifcond>
		<declaracion>
```

```
                <cuerpo_declaracion>int p=0; </cuerpo_declaracion>
            </declaracion>
            <code_block>
                <code>int r=p+2</code>
            </code_block>
            <dec_block>
                <ifcond>//bloquedeclaraciones</ifcond>
                <declaracion>
                    <cuerpo_declaracion>char*              puntero;
                    </cuerpo_declaracion>
                </declaracion>
                <dec_block>
                    <ifcond>//otrobloquedeclaraciones</ifcond>
                    <endif>endif otrobloque</endif>
                </dec_block>
                <endif>endif bloque declaraciones</endif>
            </dec_block>
            <block>
                <ifcond>if (3==3)</ifcond>
                <endif>end if</endif>
            </block>
        <endif>endif bloque codigo(primer if)</endif>
</block>
```

En este ejemplo se pueden apreciar todo lo explicado anteriormente de bloques con hijos. Tiene un bloque de código que tiene anidados por este orden “declaración”, “código”, “bloque de declaraciones” y “bloque de código”. Dentro del bloque de declaraciones anterior existe una “declaración” y otro “bloque de declaraciones”.

Para que los XML queden un poco más estéticos, para cada elemento se tiene en cuenta su nivel y usando una función auxiliar “tabuladores (int nivel)” se añaden tabuladores en función del nivel al documento XML y todo queda un poco mejor estructurado.

Una vez se han recorrido todos los elementos de la lista o cada que nos encontramos con una clase copiamos el contenido de los atributos “cabecera”, “includecadena” y “clasexml” en los ficheros XML creados.

Cuando se han recorrido todas las clases, creamos el fichero XML del proyecto general y el fichero de texto en el que guardamos el nombre de todos los ficheros XML que se han creado. Este fichero de texto es del que parte el módulo 3 para generar el código C++.

Una vez creados los ficheros XML estamos en situación de llamar al siguiente módulo, el generador de código para que procese los XML y genere el código C++ asociado a los mismos y lo guarde en los ficheros .cxx y .h correspondientes.

Ahora mostramos como es la estructura del XML que se genera para nuestro proyecto. En este caso tenemos un proyecto con una sola clase:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<file>
  <title>//Generated by Fulltick Editor v2.0 0.0000</title>
  <fltk_version>1.0106</fltk_version>
  <header_name>.h</header_name>
  <code_name>.cxx</code_name>
  <include>
    <nameinclude>FL/Fl.H</nameinclude>
    <nameinclude>"MiClase.h"</nameinclude>
  </include>
  <function>
        <type>int</type>
        <name>main</name>
        <param>
              <type>int</type>
              <name>argc</name>
        </param>
        <param>
              <type>char**</type>
              <name>argv</name>
        </param>
        <body>
              <widget>
                <type>MiClase</type>
                <name>MiClase_</name>
                <position>
                      <x>300</x>
                      <y>300</y>
                </position>
                <size>
                      <w>300</w>
                      <h>300</h>
                </size>
                <label>Proyecto estándar</label>
```

```
            </widget>
        </body>
    </function>
</file>
```

La estructura de este fichero consiste en una cabecera, el tipo de ficheros que se debe generar con el generador de código (.h y .cxx), una sección de "includes" y el código XML asociado al main.

Desde "<?xml version= " hasta "</code_name>" es lo que consideramos la cabecera y su contenido lo guardamos en el atributo "cabecera". Contiene información relativa a la codificación, la versión de la aplicación, etc.

Lo que se encuentra entre las etiquetas "<include>…</include>" son los "includes" que se deben hacer y su contenido se guarda en el atributo "includecadena". En este documento siempre habrá dos "includes", uno "FL/Fl.H" y luego habrá otro include por cada clase que tenga nuestro proyecto. En este ejemplo tenemos "Class_1.h".

La siguiente parte de este fichero XML contiene la información relativa a la función main, en la que indicamos en la sección body, los widget (en este caso los widget son las clases) que tiene nuestro proyecto. A partir de este XML se generarán un fichero ".h" y otro ".cxx" que tendrán el código C++ asociado a nuestro interfaz.

En el ejemplo se puede observar que tiene etiquetas para la posición y el tamaño. Esto se debe a que esta clase extiende de Fl_Window y esas son las coordenadas y el tamaño de la ventana.

Este fichero XML es el último archivo que se crea, dado que tenemos que incluir en él todas las clases existentes en el proyecto.

- **Generación de los ficheros XML de la calculadora**

Ahora continuando con el ejemplo de la calculadora explicado en el módulo 1, mostramos cuáles son los ficheros XML que se crean asociados a ese ejemplo. En este

ejemplo se crean tres ficheros, uno llamado "calculadora.fl", otro llamado "ClaseCalculadora.xml" y otro llamado "Calculadora.xml".

- Calculadora.fl:
  - ClaseCalculadora.xml
  - Calculadora.xml

Como se ha comentado este fichero sólo contiene el nombre de los ficheros xml que se han ido creando, y en primer lugar aparecen los ficheros XML de las clases creadas en el proyecto (en este ejemplo "ClaseCalculadora.xml") y el último es el fichero XML del main del proyecto (en este caso "Calculadora.xml").

- ClaseCalculadora.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<file>
<title>//Generated by Fulltick Editor v2.0 0.0000</title>
<fltk_version>1.0106</fltk_version>
<header_name>.h</header_name>
<code_name>.cxx</code_name>
<include>
    <nameinclude>FL/Fl.H</nameinclude>
    <nameinclude>FL/Fl_Window.H</nameinclude>
    <nameinclude>FL/Fl_Button.H</nameinclude>
    <nameinclude>FL/Fl_Multiline_Output.H</nameinclude>
    <nameincludeuser><stdlib.h></nameincludeuser>
    <nameincludeuser><iostream></nameincludeuser>
</include>
<class>
<name>ClaseCalculadora</name>
<extends>Fl_Window</extends>
<label>Calculadora</label>
<position>
  <x>474</x>
  <y>94</y>
</position>
<size>
  <w>400</w>
  <h>500</h>
</size>
<label_font>0</label_font>
<label_size>14</label_size>
<label_color>56</label_color>
<label_align>1</label_align>
<label_align_index>0</label_align_index>
<box>2</box>
```

```
<box_color>49</box_color>

<attribute>
  <type>Fl_Button*</type>
  <name>boton0</name>
  <visibility>public</visibility>
  <parent>ClaseCalculadora</parent>
  <label>0</label>
  <position>
    <x>100</x>
    <y>400</y>
  </position>
  <size>
    <w>100</w>
    <h>100</h>
  </size>
  <label_font>0</label_font>
  <label_size>50</label_size>
  <label_color>56</label_color>
  <label_align>0</label_align>
  <label_align_index>0</label_align_index>
  <label_type>FL_NORMAL_LABEL</label_type>
  <label_type_index>1</label_type_index>
  <box>2</box>
  <box_color>93</box_color>
  <down_box>3</down_box>
  <down_box_color>49</down_box_color>
  <callback>
    <event>FL_RELEASE1</event>
    <call>boton0_left_cb()</call>
  </callback>
  </attribute>

<function>
  <name>boton0_left_cb</name>
  <visibility>public</visibility>
  <body>
        <code_block>imprime("0");</code_block>
  </body>
</function>

<attribute>
  <type>Fl_Button*</type>
  <name>boton1</name>
  <visibility>public</visibility>
  <parent>ClaseCalculadora</parent>
  <label>1</label>
  <position>
    <x>0</x>
    <y>300</y>
  </position>
  <size>
    <w>100</w>
    <h>100</h>
  </size>
```

```
  <label_font>0</label_font>
   <label_size>50</label_size>
   <label_color>56</label_color>
   <label_align>0</label_align>
   <label_align_index>0</label_align_index>
   <label_type>FL_NORMAL_LABEL</label_type>
   <label_type_index>1</label_type_index>
   <box>2</box>
   <box_color>93</box_color>
   <down_box>3</down_box>
   <down_box_color>49</down_box_color>
   <callback>
     <event>FL_RELEASE1</event>
     <call>boton1_left_cb()</call>
  </callback>
</attribute>

<function>
   <name>boton1_left_cb</name>
   <visibility>public</visibility>
   <body>
         <code_block>imprime("1");</code_block>
  </body>
</function>

<attribute>
   <type>Fl_Button*</type>
   <name>boton2</name>
   <visibility>public</visibility>
   <parent>ClaseCalculadora</parent>
   <label>2</label>
   <position>
     <x>100</x>
     <y>300</y>
  </position>
   <size>
     <w>100</w>
     <h>100</h>
  </size>
   <label_font>0</label_font>
   <label_size>50</label_size>
   <label_color>56</label_color>
   <label_align>0</label_align>
   <label_align_index>0</label_align_index>
   <label_type>FL_NORMAL_LABEL</label_type>
   <label_type_index>1</label_type_index>
   <box>2</box>
   <box_color>93</box_color>
   <down_box>3</down_box>
   <down_box_color>49</down_box_color>
   <callback>
     <event>FL_RELEASE1</event>
     <call>boton2_left_cb()</call>
  </callback>
</attribute>
```

```
<function>
  <name>boton2_left_cb</name>
  <visibility>public</visibility>
  <body>
        <code_block>imprime("2");</code_block>
  </body>
</function>

<attribute>
  <type>Fl_Button*</type>
  <name>boton3</name>
  <visibility>public</visibility>
  <parent>ClaseCalculadora</parent>
  <label>3</label>
  <position>
    <x>200</x>
    <y>300</y>
  </position>
  <size>
    <w>100</w>
    <h>100</h>
  </size>
  <label_font>0</label_font>
  <label_size>50</label_size>
  <label_color>56</label_color>
  <label_align>0</label_align>
  <label_align_index>0</label_align_index>
  <label_type>FL_NORMAL_LABEL</label_type>
  <label_type_index>1</label_type_index>
  <box>2</box>
  <box_color>93</box_color>
  <down_box>3</down_box>
  <down_box_color>49</down_box_color>
  <callback>
    <event>FL_RELEASE1</event>
    <call>boton3_left_cb()</call>
  </callback>
</attribute>

<function>
  <name>boton3_left_cb</name>
  <visibility>public</visibility>
  <body>
        <code_block>imprime("3");</code_block>
  </body>
</function>

<attribute>
  <type>Fl_Button*</type>
  <name>boton4</name>
  <visibility>public</visibility>
  <parent>ClaseCalculadora</parent>
  <label>4</label>
  <position>
```

```
        <x>0</x>
        <y>200</y>
      </position>
      <size>
        <w>100</w>
        <h>100</h>
      </size>
      <label_font>0</label_font>
      <label_size>50</label_size>
      <label_color>56</label_color>
      <label_align>0</label_align>
      <label_align_index>0</label_align_index>
      <label_type>FL_NORMAL_LABEL</label_type>
      <label_type_index>1</label_type_index>
      <box>2</box>
      <box_color>93</box_color>
      <down_box>3</down_box>
      <down_box_color>49</down_box_color>
      <callback>
        <event>FL_RELEASE1</event>
        <call>boton4_left_cb()</call>
      </callback>
    </attribute>

    <function>
      <name>boton4_left_cb</name>
      <visibility>public</visibility>
      <body>
            <code_block>imprime("4");</code_block>
      </body>
      </function>

- <attribute>
      <type>Fl_Button*</type>
      <name>boton5</name>
      <visibility>public</visibility>
      <parent>ClaseCalculadora</parent>
      <label>5</label>
      <position>
        <x>100</x>
        <y>200</y>
      </position>
      <size>
        <w>100</w>
        <h>100</h>
      </size>
      <label_font>0</label_font>
      <label_size>50</label_size>
      <label_color>56</label_color>
      <label_align>0</label_align>
      <label_align_index>0</label_align_index>
      <label_type>FL_NORMAL_LABEL</label_type>
      <label_type_index>1</label_type_index>
      <box>2</box>
      <box_color>93</box_color>
```

```
    <down_box>3</down_box>
    <down_box_color>49</down_box_color>
    <callback>
      <event>FL_RELEASE1</event>
      <call>boton5_left_cb()</call>
    </callback>
  </attribute>

  <function>
    <name>boton5_left_cb</name>
    <visibility>public</visibility>
    <body>
          <code_block>imprime("5");</code_block>
    </body>
  </function>

  <attribute>
    <type>Fl_Button*</type>
    <name>boton6</name>
    <visibility>public</visibility>
    <parent>ClaseCalculadora</parent>
    <label>6</label>
    <position>
      <x>200</x>
      <y>200</y>
    </position>
    <size>
      <w>100</w>
      <h>100</h>
    </size>
    <label_font>0</label_font>
    <label_size>50</label_size>
    <label_color>56</label_color>
    <label_align>0</label_align>
    <label_align_index>0</label_align_index>
    <label_type>FL_NORMAL_LABEL</label_type>
    <label_type_index>1</label_type_index>
    <box>2</box>
    <box_color>93</box_color>
    <down_box>3</down_box>
    <down_box_color>49</down_box_color>
    <callback>
      <event>FL_RELEASE1</event>
      <call>boton6_left_cb()</call>
    </callback>
  </attribute>

  <function>
    <name>boton6_left_cb</name>
    <visibility>public</visibility>
    <body>
          <code_block>imprime("6");</code_block>
    </body>
  </function>
```

```
<attribute>
  <type>Fl_Button*</type>
  <name>boton7</name>
  <visibility>public</visibility>
  <parent>ClaseCalculadora</parent>
  <label>7</label>
  <position>
    <x>0</x>
    <y>100</y>
  </position>
  <size>
    <w>100</w>
    <h>100</h>
  </size>
  <label_font>0</label_font>
  <label_size>50</label_size>
  <label_color>56</label_color>
  <label_align>0</label_align>
  <label_align_index>0</label_align_index>
  <label_type>FL_NORMAL_LABEL</label_type>
  <label_type_index>1</label_type_index>
  <box>2</box>
  <box_color>93</box_color>
  <down_box>3</down_box>
  <down_box_color>49</down_box_color>
  <callback>
    <event>FL_RELEASE1</event>
    <call>boton7_left_cb()</call>
  </callback>
</attribute>

<function>
  <name>boton7_left_cb</name>
  <visibility>public</visibility>
  <body>
        <code_block>imprime("7");</code_block>
  </body>
</function>

<attribute>
  <type>Fl_Button*</type>
  <name>boton8</name>
  <visibility>public</visibility>
  <parent>ClaseCalculadora</parent>
  <label>8</label>
  <position>
    <x>100</x>
    <y>100</y>
  </position>
  <size>
    <w>100</w>
    <h>100</h>
  </size>
  <label_font>0</label_font>
  <label_size>50</label_size>
```

```
    <label_color>56</label_color>
    <label_align>0</label_align>
    <label_align_index>0</label_align_index>
    <label_type>FL_NORMAL_LABEL</label_type>
    <label_type_index>1</label_type_index>
    <box>2</box>
    <box_color>93</box_color>
    <down_box>3</down_box>
    <down_box_color>49</down_box_color>
    <callback>
      <event>FL_RELEASE1</event>
      <call>boton8_left_cb()</call>
    </callback>
  </attribute>

<function>
  <name>boton8_left_cb</name>
  <visibility>public</visibility>
  <body>
        <code_block>imprime("8");</code_block>
  </body>
</function>

<attribute>
  <type>Fl_Button*</type>
  <name>boton9</name>
  <visibility>public</visibility>
  <parent>ClaseCalculadora</parent>
  <label>9</label>
  <position>
    <x>200</x>
    <y>100</y>
  </position>
  <size>
    <w>100</w>
    <h>100</h>
  </size>
  <label_font>0</label_font>
  <label_size>50</label_size>
  <label_color>56</label_color>
  <label_align>0</label_align>
  <label_align_index>0</label_align_index>
  <label_type>FL_NORMAL_LABEL</label_type>
  <label_type_index>1</label_type_index>
  <box>2</box>
  <box_color>93</box_color>
  <down_box>3</down_box>
  <down_box_color>49</down_box_color>
  <callback>
    <event>FL_RELEASE1</event>
    <call>boton9_left_cb()</call>
  </callback>
</attribute>

<function>
```

```
 <name>boton9_left_cb</name>
 <visibility>public</visibility>
 <body>
       <code_block>imprime("9");</code_block>
 </body>
</function>

<function>
 <name>imprime</name>
 <type>void</type>
 <visibility>public</visibility>
 <param>char* cadena</param>
 <body>
      <code_block>if(accion == 1){ borrar(); accion = 2; } char* numero;
    numero = (char*)output->value(); numero = concatenar(numero,
    cadena); output->value(numero);</code_block>
 </body>
</function>

<function>
 <name>concatenar</name>
 <type>char*</type>
 <visibility>public</visibility>
 <param>char* s1, char* s2</param>
 <body>
    <code_block>//Función que concatena las cadenas s1 y s2; //Este método
    es poco eficiente, sólo vale para cadenas pequeñas int l1 = strlen(s1); int
    l2 = strlen(s2); char* res = (char*)malloc((l1+l2+1)*sizeof(char)); for(int
    i=0; i<l1; i++){ res[i]=s1[i]; } for(int j=0; j<l2; j++){ res[l1+j]=s2[j]; }
    res[l1+l2] = '\0';//Importante el '\0' return res;</code_block>
 </body>
</function>

 <attribute>
    <type>Fl_Multiline_Output*</type>
    <name>output</name>
    <visibility>public</visibility>
    <parent>ClaseCalculadora</parent>
    <position>
     <x>0</x>
     <y>0</y>
    </position>
    <size>
      <w>400</w>
      <h>100</h>
    </size>
    <label_font>5</label_font>
    <label_size>14</label_size>
    <label_color>56</label_color>
    <label_align>0</label_align>
    <label_align_index>0</label_align_index>
    <label_type>FL_NORMAL_LABEL</label_type>
    <label_type_index>1</label_type_index>
    <box>2</box>
    <box_color>7</box_color>
```

```
    <text_font>5</text_font>
    <text_size>50</text_size>
    <text_color>0</text_color>
  </attribute>

 <attribute>
  <type>Fl_Button*</type>
  <name>botonMas</name>
  <visibility>public</visibility>
  <parent>ClaseCalculadora</parent>
  <label>+</label>
  <position>
    <x>300</x>
    <y>100</y>
  </position>
  <size>
    <w>100</w>
    <h>100</h>
  </size>
  <label_font>1</label_font>
  <label_size>50</label_size>
  <label_color>56</label_color>
  <label_align>0</label_align>
  <label_align_index>0</label_align_index>
  <label_type>FL_NORMAL_LABEL</label_type>
  <label_type_index>1</label_type_index>
  <box>2</box>
  <box_color>183</box_color>
  <down_box>3</down_box>
  <down_box_color>49</down_box_color>
  <tooltip>Suma</tooltip>
  <callback>
    <event>FL_RELEASE1</event>
    <call>botonMas_left_cb()</call>
  </callback>
  </attribute>

  <function>
    <name>botonMas_left_cb</name>
    <visibility>public</visibility>
    <body>
        <code_block>operar('+');</code_block>
    </body>
  </function>

  <attribute>
    <type>Fl_Button*</type>
    <name>botonMenos</name>
    <visibility>public</visibility>
    <parent>ClaseCalculadora</parent>
    <label>-</label>
    <position>
      <x>300</x>
      <y>200</y>
```

```
  </position>
   <size>
     <w>100</w>
     <h>100</h>
  </size>
  <label_font>1</label_font>
   <label_size>50</label_size>
   <label_color>56</label_color>
   <label_align>0</label_align>
   <label_align_index>0</label_align_index>
   <label_type>FL_NORMAL_LABEL</label_type>
   <label_type_index>1</label_type_index>
   <box>2</box>
   <box_color>183</box_color>
   <down_box>3</down_box>
   <down_box_color>49</down_box_color>
   <tooltip>Resta</tooltip>
   <callback>
     <event>FL_RELEASE1</event>
     <call>botonMenos_left_cb()</call>
  </callback>
</attribute>

<function>
   <name>botonMenos_left_cb</name>
   <visibility>public</visibility>
   <body>
         <code_block>operar('-');</code_block>
  </body>
</function>

<attribute>
   <type>Fl_Button*</type>
   <name>botonMul</name>
   <visibility>public</visibility>
   <parent>ClaseCalculadora</parent>
   <label>*</label>
   <position>
     <x>300</x>
     <y>300</y>
  </position>
   <size>
     <w>100</w>
     <h>100</h>
  </size>
  <label_font>1</label_font>
   <label_size>50</label_size>
   <label_color>56</label_color>
   <label_align>0</label_align>
   <label_align_index>0</label_align_index>
   <label_type>FL_NORMAL_LABEL</label_type>
   <label_type_index>1</label_type_index>
   <box>2</box>
   <box_color>183</box_color>
```

```
    <down_box>3</down_box>
    <down_box_color>49</down_box_color>
    <tooltip>Multiplicación</tooltip>
    <callback>
      <event>FL_RELEASE1</event>
      <call>botonMul_left_cb()</call>
    </callback>
  </attribute>

  <function>
    <name>botonMul_left_cb</name>
    <visibility>public</visibility>
    <body>
          <code_block>operar('*');</code_block>
    </body>
  </function>

  <attribute>
    <type>Fl_Button*</type>
    <name>botonDiv</name>
    <visibility>public</visibility>
    <parent>ClaseCalculadora</parent>
    <label>/</label>
    <position>
       <x>300</x>
       <y>400</y>
    </position>
    <size>
      <w>100</w>
      <h>100</h>
    </size>
    <label_font>1</label_font>
    <label_size>50</label_size>
    <label_color>56</label_color>
    <label_align>0</label_align>
    <label_align_index>0</label_align_index>
    <label_type>FL_NORMAL_LABEL</label_type>
    <label_type_index>1</label_type_index>
    <box>2</box>
    <box_color>183</box_color>
    <down_box>3</down_box>
    <down_box_color>49</down_box_color>
    <tooltip>División</tooltip>
    <callback>
       <event>FL_RELEASE1</event>
       <call>botonDiv_left_cb()</call>
    </callback>
  </attribute>

  <function>
    <name>botonDiv_left_cb</name>
    <visibility>public</visibility>
    <body>
          <code_block>operar('/');</code_block>
```

```
  </body>
</function>

<attribute>
  <type>Fl_Button*</type>
  <name>botonIgual</name>
  <visibility>public</visibility>
  <parent>ClaseCalculadora</parent>
  <label>=</label>
  <position>
    <x>200</x>
    <y>400</y>
  </position>
  <size>
    <w>100</w>
    <h>100</h>
  </size>
  <label_font>5</label_font>
  <label_size>50</label_size>
  <label_color>56</label_color>
  <label_align>0</label_align>
  <label_align_index>0</label_align_index>
  <label_type>FL_NORMAL_LABEL</label_type>
  <label_type_index>1</label_type_index>
  <box>2</box>
  <box_color>79</box_color>
  <down_box>3</down_box>
  <down_box_color>49</down_box_color>
  <tooltip>Resultado</tooltip>
  <callback>
    <event>FL_RELEASE1</event>
    <call>botonIgual_left_cb()</call>
  </callback>
  </attribute>

<function>
  <name>botonIgual_left_cb</name>
  <visibility>public</visibility>
  <body>
        <code_block>resolver();</code_block>
  </body>
</function>

<attribute>
  <type>Fl_Button*</type>
  <name>botonC</name>
  <visibility>public</visibility>
  <parent>ClaseCalculadora</parent>
  <label>C</label>
  <position>
    <x>0</x>
    <y>400</y>
  </position>
  <size>
```

```
      <w>100</w>
      <h>100</h>
    </size>
    <label_font>5</label_font>
    <label_size>50</label_size>
    <label_color>56</label_color>
    <label_align>0</label_align>
    <label_align_index>0</label_align_index>
    <label_type>FL_NORMAL_LABEL</label_type>
    <label_type_index>1</label_type_index>
    <box>2</box>
    <box_color>79</box_color>
    <down_box>3</down_box>
    <down_box_color>49</down_box_color>
    <tooltip>Borrar</tooltip>
    <callback>
       <event>FL_RELEASE1</event>
       <call>botonC_left_cb()</call>
    </callback>
</attribute>

<function>
    <name>botonC_left_cb</name>
    <visibility>public</visibility>
    <body>
          <code_block>inicializar();</code_block>
    </body>
</function>

<function>
    <name>operar</name>
    <type>void</type>
    <visibility>public</visibility>
    <param>char c</param>
    <body>
          <code_block>op1 = atoi(output->value()); accion = 1; operacion =
          c;</code_block>
    </body>
</function>

<function>
    <name>resolver</name>
    <type>void</type>
    <visibility>public</visibility>
    <body>
      <code_block>char resultado[20]; op2 = atoi(output->value());
      switch(operacion){ case '+': itoa(op1 + op2, resultado, 10); output-
      >value(resultado); break; case '-': itoa(op1 - op2, resultado, 10); output-
      >value(resultado); break; case '*': itoa(op1 * op2, resultado, 10);
      output->value(resultado); break; case '/': if(op2 == 0){ output-
      >value("ERROR"); } else{ itoa(op1 / op2, resultado, 10); output-
      >value(resultado); } break; default: output->value("ERROR");
      }</code_block>
    </body>
</function>
```

```
<function>
  <name>borrar</name>
  <type>void</type>
  <visibility>public</visibility>
  <body>
        <code_block>output->value("");</code_block>
  </body>
</function>

<function>
  <name>inicializar</name>
  <type>void</type>
  <visibility>public</visibility>
  <body>
        <code_block>borrar(); accion = 0;</code_block>
  </body>
</function>

<declaracion>int op1, op2;</declaracion>
<declaracion>char operacion;</declaracion>
<declaracion>int accion;//0->escribimos primer operando, 1->pulsamos
  operación, 2-> escribimos segundo operando</declaracion>
  </class>
</file>
```

Ahora explicamos a grandes rasgos el código que se ha generado.

En primer lugar tenemos la información relativa a la versión del editor, los incluyes que se hacen y el código asociado a la clase que extiende a ventana:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<file>
<title>//Generated by Fulltick Editor v2.0 0.0000</title>
<fltk_version>1.0106</fltk_version>
<header_name>.h</header_name>
<code_name>.cxx</code_name>
<include>
    <nameinclude>FL/Fl.H</nameinclude>
    <nameinclude>FL/Fl_Window.H</nameinclude>
    <nameinclude>FL/Fl_Button.H</nameinclude>
    <nameinclude>FL/Fl_Multiline_Output.H</nameinclude>
    <nameincludeuser><stdlib.h></nameincludeuser>
    <nameincludeuser><iostream></nameincludeuser>
</include>
<class>
<name>ClaseCalculadora</name>
<extends>Fl_Window</extends>
<label>Calculadora</label>
<position>
  <x>474</x>
```

```
    <y>94</y>
  </position>
  <size>
    <w>400</w>
    <h>500</h>
  </size>
  <label_font>0</label_font>
  <label_size>14</label_size>
  <label_color>56</label_color>
  <label_align>1</label_align>
  <label_align_index>0</label_align_index>
  <box>2</box>
  <box_color>49</box_color>
```

En la sección de includes hay dos entre las etiquetas `<nameincludeuser>` y esto se debe a las declaraciones que ha hecho el usuario manualmente y que dado que contienen la cadena "#include" la aplicación los guarda en esta sección. El resto de includes son los asociados a los botones, cuadros de texto y demás elementos de la aplicación.

A continuación viene el código asociado a los números de la calculadora los cuales tienen todos un aspecto similar:

```
<attribute>
    <type>Fl_Button*</type>
    <name>boton0</name>
    <visibility>public</visibility>
    <parent>ClaseCalculadora</parent>
    <label>0</label>
    <position>
      <x>100</x>
      <y>400</y>
    </position>
    <size>
      <w>100</w>
      <h>100</h>
    </size>
    <label_font>0</label_font>
    <label_size>50</label_size>
    <label_color>56</label_color>
    <label_align>0</label_align>
    <label_align_index>0</label_align_index>
    <label_type>FL_NORMAL_LABEL</label_type>
    <label_type_index>1</label_type_index>
    <box>2</box>
    <box_color>93</box_color>
    <down_box>3</down_box>
    <down_box_color>49</down_box_color>
    <callback>
      <event>FL_RELEASE1</event>
```

```
      <call>boton0_left_cb()</call>
    </callback>
  </attribute>

  <function>
    <name>boton0_left_cb</name>
    <visibility>public</visibility>
    <body>
          <code_block>imprime("0");</code_block>
    </body>
  </function>
```

Aquí se especifican las características de los botones tales como el tipo, el nombre, la visibilidad, su padre(que en este caso es la ClaseCalculadora), la etiqueta,su posición, tamaño, las fuentes, colores,etc. Cada botón tiene asociado un callback, que implica la ejecución de la función "NombreBoton_left_Cb()" y consiste en cuando el usuario pulse un número con el botón izquierdo del ratón se ejecutará la función "imprime" que imprime en el cuadro de texto el número que se ha pulsado.

Después tenemos las funciones "imprime" y "concatenar":

```
<function>
  <name>imprime</name>
  <type>void</type>
  <visibility>public</visibility>
  <param>char* cadena</param>
  <body>
      <code_block>if(accion == 1){ borrar(); accion = 2; } char* numero;
    numero = (char*)output->value(); numero = concatenar(numero,
    cadena); output->value(numero);</code_block>
  </body>
 </function>

 <function>
  <name>concatenar</name>
  <type>char*</type>
  <visibility>public</visibility>
  <param>char* s1, char* s2</param>
  <body>
    <code_block>//Función que concatena las cadenas s1 y s2; //Este método
    es poco eficiente, sólo vale para cadenas pequeñas int l1 = strlen(s1); int
    l2 = strlen(s2); char* res = (char*)malloc((l1+l2+1)*sizeof(char)); for(int
    i=0; i<l1; i++){ res[i]=s1[i]; } for(int j=0; j<l2; j++){ res[l1+j]=s2[j]; }
    res[l1+l2] = '\0';//Importante el '\0' return res;</code_block>
  </body>
 </function>
```

En el XML sobre estas funciones escribimos su nombre, el tipo de lo que devuelven, la visibilidad, los argumentos y también tienen un bloque de código dentro que indican la acción a realizar por esta función.

Tras estas funciones en el XML nos encontramos con el campo de texto en el que iremos viendo lo que escribe el usuario y los botones de las operaciones. Estos elementos tienen un código XML muy parecido al de los botones de los números comentados anteriormente, con la diferencia que el campo de texto no tiene ningún callback y los botones de las operaciones tienen asociado un callback que llama a la función "operar".

También tenemos los botones igual y C que siguen la misma estructura de todos los botones, pero que tienen como callbacks la llamada a las funciones "resolver" e "inicializar" respectivamente.

Por último nos encontramos con la declaración de unas cuantas variables de la clase:

```
<declaracion>int op1, op2;</declaracion>
<declaracion>char operacion;</declaracion>
<declaracion>int accion;//0->escribimos primer operando, 1->pulsamos operación, 2-> escribimos segundo operando</declaracion>
```

- calculadora.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<file>
<title>//Generated by Fulltick Editor v2.0 0.0000</title>
<fltk_version>1.0106</fltk_version>
<header_name>.h</header_name>
<code_name>.cxx</code_name>
<include>
  <nameinclude>FL/Fl.H</nameinclude>
  <nameinclude>"ClaseCalculadora.h"</nameinclude>
</include>
<function>
  <type>int</type>
  <name>main</name>
  <param>
      <type>int</type>
      <name>argc</name>
  </param>
```

```
<param>
        <type>char**</type>
        <name>argv</name>
</param>
<body>
        <widget>
                <type>ClaseCalculadora</type>
                <name>ClaseCalculadora_</name>
                <position>
                  <x>474</x>
                  <y>94</y>
                </position>
                <size>
                  <w>400</w>
                  <h>500</h>
                </size>
                <label>Calculadora</label>
        </widget>
</body>
</function>
</file>
```

En este fichero se puede ver que está la sección dedicada a la versión de la aplicación y el tipo de ficheros que se debe generar con el generador de código (.h y .cxx). En la sección de includes tenemos “FL/Fl.H” que lo tendrán todos los proyectos, así como el include de la clase (ClaseCalculadora). El resto del contenido de este fichero es el código para la función “main”, en el que habrá una sección “widget” por cada clase que exista en nuestro proyecto.

7.3 Módulo 3: Generador de Código

El tercer módulo de nuestro editor gráfico de interfaces gráficas se ejecuta cuando pulsamos sobre Archivo → Escribir código, recibe como entrada un fichero de proyecto creado por el segundo módulo y produce como salida una serie de ficheros de código y de declaraciones en código C++ listo para compilar y ejecutar.

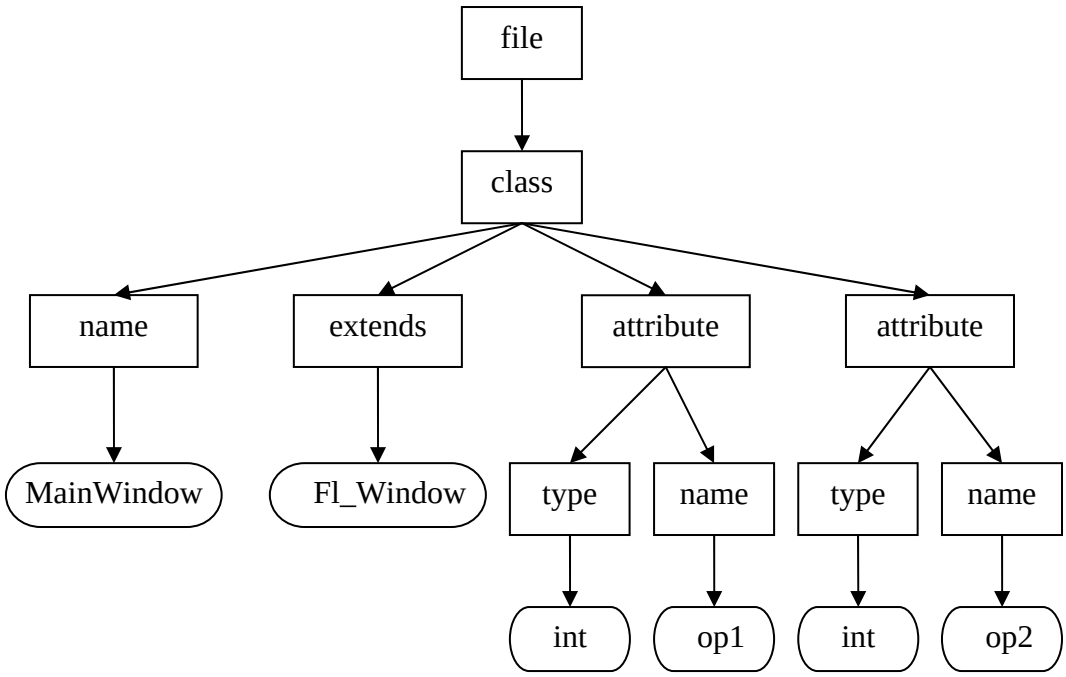
La entrada que recibe este módulo es un fichero de proyecto que contiene la ruta de un cierto número de ficheros XML, uno en cada línea. El procesamiento de estos ficheros XML se lleva a cabo usando la librería Xerces-C versión 2.7.0, cuyo funcionamiento fue expicado anteriormente.

En nuestro caso nos valemos de la librería Xerces para la lectura de ficheros XML, para esto vamos a usar el DOMBuilder, que consiste en un parser de XML que genera una estructura de árbol donde almacena la información contenida en el fichero XML. Vamos a poner un ejemplo del funcionamiento del DOMBuilder para comprender mejor su forma de operar:

Supongamos el siguiente fichero de entrada escrito en XML:

```
<file>
        <class>
                <name>MainWindow</name>
                <extends>Fl_Window</extends>
                <attribute>
                        <type>int</type>
                        <name>op1</name>
                </attribute>
                <attribute>
                        <type>int</type>
                        <name>op2</name>
                </attribute>
        </class>
</file>
```

Al procesar este fichero con el DOMBuilder obtenemos una estructura en forma de árbol como la siguiente:



Ya conocemos el funcionamiento de la librería Xerces, ahora vamos a proceder a explicar con más detalle la forma de operar de este módulo.

Cuando pulsamos sobre Archivo → Escribir código se generan una serie de estructuras para generar el código. Una de estas estructuras es la lista de constructores. A partir de un fichero XML donde están guardados todos los constructores de los widgets que se pueden añadir con nuestro editor se genera una lista de constructores. Esta lista se usa cuando nos creamos con el editor una clase que extiende a otra anteriormente creada o a un widget de la librería FLTK. Cuando ocurre esto recorremos la lista para buscar todos los constructores de la clase que se extiende y nos creamos, por cada constructor de la superclase, un constructor que llama al constructor de la superclase y que contiene el código que crea un objeto de nuestra clase.

Tras esto se llama a una función que lee el fichero de proyecto (.fl), donde están listados todos los ficheros XML que componen el proyecto, uno por cada línea. Para cada línea se llama a la función procesarFichero(char* xmlFile), a la que le pasamos la ruta del fichero XML que queremos procesar. Esta función está contenida en el fichero generador.cpp y su misión es inicializar todo el motor del Xerces y llamar a la función "generaCodigo", a la que le pasa la raiz del árbol DOM generado.

La función “generaCodigo” va recorriendo los hijos del árbol mirando sus etiquetas, si por ejemplo encuentra la etiqueta “class” se llama a la función “generaCodigoClass”, a la que le pasa el nodo del hijo que tiene la etiqueta “class” y que a su vez recorre a todos sus hijos y va llamando a otras funciones dependiendo de las etiquetas que tengan los mismos.

Durante el recorrido del árbol se va generando el código C++ en ciertos puntos, vamos a mostrar el código que se generaría si nos encontráramos con el código que vimos en el ejemplo del árbol DOM. El XML sería el siguiente:

```
<file>
        <class>
                <name>MainWindow</name>
                <extends>Fl_Window</extends>
                <attribute>
                        <type>int</type>
                        <name>op1</name>
                </attribute>
                <attribute>
                        <type>int</type>
                        <name>op2</name>
                </attribute>
        </class>
</file>
```

Para este ejemplo se realizarían las siguientes acciones: se llamaría a generaCodigo pasándole el nodo que tiene la etiqueta “file”, “generaCodigo” recorrería sus hijos encontrando la etiqueta “class” y llamaría a la función “generaCodigoClass”, dentro de esta función se empezaría la formación del archivo .h, en el que pondríamos la cadena “class”, después nos encontraríamos la etiqueta “name”, leeríamos el nombre de la clase y lo concatenaríamos con lo que ya teníamos en el archivo .h, por lo que el contenido actual sería “class MainWindow”. Después nos encontraríamos con la etiqueta “extends” y concatenaríamos la cadena “:public Fl_Window” al contenido del archivo .h. Seguiríamos así hasta recorrer todo el árbol, de modo que al final el contenido del archivo .h sería el siguiente:

```
class MainWindow : public Fl_Window{
        public:
                int op1;
                int op2;
};
```

Cada vez que entramos en la función “generaCodigo” se inicializan una serie de variables que van a contener datos de utilidad a la hora de generar los ficheros de código, y al final de la función se liberan. Vamos a nombrar algunas de las más significativas:

- headerFile1: Contiene el código del fichero de cabeceras que va desde el principio hasta la parte de declaraciones private (inclusive).
- headerFile2: Contiene la parte de declaraciones protected del fichero de cabeceras.
- headerFile3: Contiene el código del fichero de cabeceras que va desde la parte de declaraciones public (inclusive) hasta el final.
- codeFile1: Contiene los comentarios iniciales e includes del fichero de código.
- codeFile2: Contiene las funciones que no pertenecen a la clase dentro del fichero de código.
- codeFile3: Contiene las funciones que pertenecen a la clase dentro del fichero de código.
- codeFile4: Parte del fichero de código que contiene los constructores.
- eventFile: Código de una clase que contiene las funciones estáticas asociadas a los eventos.

A medida que recorremos el fichero XML se van rellenando estas variables para ser volcadas en los ficheros de cabeceras (con nombre “NombreClase.h”) y de código (llamado “NombreClase.cxx”). Si alguno de los elementos que componen el proyecto tiene algún callback asociado también se crea un fichero llamado “NombreClase_e.h” donde existe una clase llamada “NombreClase_e” que contiene todas las funciones estáticas asociadas a estos eventos.

- **Generación del código C++ de la calculadora**

Siguiendo con el ejemplo visto en los módulos anteriores vamos a proceder a explicar la forma en la que este módulo genera el código C++ de la calculadora, tomando como entrada los ficheros XML generados en el módulo 2. Tenemos 1 fichero de proyecto llamado "calculadora.fl" y dos ficheros XML ("calculadora.xml" y "ClaseCalculadora.xml"). Esto generaría 5 ficheros: "calculadora.h", "calculadora.cxx", "ClaseCalculadora.h", "ClaseCalculadora.cxx" y "ClaseCalculadora_e.h". Primero vamos a ver el código C++ de estos ficheros y después procederemos a explicarlos más detalladamente:

- Archivo "calculadora.h"

```
//Generated by Fulltick Editor v2.0 0.0000

#ifndef calculadora_h
#define calculadora_h

#include <FL/Fl.H>
#include "ClaseCalculadora.h"

#endif
```

Este archivo contiene únicamente 2 includes, el primero para poder hacer el Fl::run() en el main y el segundo para poder crear la calculadora.

- Archivo "calculadora.cxx"

```
//Generated by Fulltick Editor v2.0 0.0000

#include "calculadora.h"

int main(int argc, char** argv){
    ClaseCalculadora* ClaseCalculadora_ = new
ClaseCalculadora(474,94,400,500,"Calculadora");
```

```
    ClaseCalculadora_->show(argc, argv);
    return Fl::run();
}
```

Este archivo contiene el include del .h y un main que se crea por defecto en el que se meten todos los widgets que aparecen en el fichero XML, en este caso solo uno, que es la calculadora, creada con la posición, tamaño y etiqueta especificadas por el usuario.

- Archivo “ClaseCalculadora.h”

```
//Generated by Fulltick Editor v2.0 0.0000

#ifndef ClaseCalculadora_h
#define ClaseCalculadora_h

#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Multiline_Output.H>
#include <stdlib.h>
#include <iostream>

class ClaseCalculadora : public Fl_Window{
public:
Fl_Button* boton0;
void boton0_left_cb();
Fl_Button* boton1;
void boton1_left_cb();
Fl_Button* boton2;
void boton2_left_cb();
Fl_Button* boton3;
void boton3_left_cb();
Fl_Button* boton4;
```

```
void boton4_left_cb();
Fl_Button* boton5;
void boton5_left_cb();
Fl_Button* boton6;
void boton6_left_cb();
Fl_Button* boton7;
void boton7_left_cb();
Fl_Button* boton8;
void boton8_left_cb();
Fl_Button* boton9;
void boton9_left_cb();
void imprime(char* cadena);
char* concatenar(char* s1, char* s2);
Fl_Multiline_Output* output;
Fl_Button* botonMas;
void botonMas_left_cb();
Fl_Button* botonMenos;
void botonMenos_left_cb();
Fl_Button* botonMul;
void botonMul_left_cb();
Fl_Button* botonDiv;
void botonDiv_left_cb();
Fl_Button* botonIgual;
void botonIgual_left_cb();
Fl_Button* botonC;
void botonC_left_cb();
void operar(char c);
void resolver();
void borrar();
void inicializar();
ClaseCalculadora(int x, int y, int w, int h);
ClaseCalculadora(int x, int y, int w, int h, char* label);
ClaseCalculadora(int w, int h);
```

```
};

#endif
```

En este fichero lo primero que aparece son todos los includes necesarios para poder crear los botones, campo de texto, etc que componen la calculadora y otros más puestos a mano por el usuario para poder hacer uso de ciertas librerías en su código. Después aparece la declaración de una clase (“ClaseCalculadora”) que extiende a Fl_Window y que contiene una serie de atributos. Estos atributos son todos los widgets que componen la interfaz de la calculadora. También aparecen las funciones asociadas a cada uno de los callbacks de los botones y otras funciones que vamos a necesitar para programar el funcionamiento de la calculadora. Al final del fichero aparecen tres constructores que se han generado por defecto. Estos constructores se corresponden con los tres constructores que tiene la clase Fl_Window, y son incorporados automáticamente por la herramienta.

- Archivo “ClaseCalculadora.cxx”

```
//Generated by Fulltick Editor v2.0 0.0000

#include "ClaseCalculadora.h"
#include "ClaseCalculadora_e.h"

int op1, op2;
char operacion;
int accion;//0->escribimos primer operando, 1->pulsamos operación, 2-> escribimos segundo operando
void ClaseCalculadora::boton0_left_cb(){
imprime("0");
}

void ClaseCalculadora::boton1_left_cb(){
imprime("1");
}
```

```
void ClaseCalculadora::boton2_left_cb(){
imprime("2");
}

void ClaseCalculadora::boton3_left_cb(){
imprime("3");
}

void ClaseCalculadora::boton4_left_cb(){
imprime("4");
}

void ClaseCalculadora::boton5_left_cb(){
imprime("5");
}

void ClaseCalculadora::boton6_left_cb(){
imprime("6");
}

void ClaseCalculadora::boton7_left_cb(){
imprime("7");
}

void ClaseCalculadora::boton8_left_cb(){
imprime("8");
}

void ClaseCalculadora::boton9_left_cb(){
imprime("9");
}

void ClaseCalculadora::imprime(char* cadena){
```

```
if(accion == 1){
        borrar();
        accion = 2;
}
char* numero;
numero = (char*)output->value();
numero = concatenar(numero, cadena);
output->value(numero);
}

char* ClaseCalculadora::concatenar(char* s1, char* s2){
//Función que concatena las cadenas s1 y s2;
//Este método es poco eficiente, sólo vale para cadenas pequeñas
int l1 = strlen(s1);
int l2 = strlen(s2);
char* res = (char*)malloc((l1+l2+1)*sizeof(char));
for(int i=0; i<l1; i++){
        res[i]=s1[i];
}
for(int j=0; j<l2; j++){
        res[l1+j]=s2[j];
}
res[l1+l2] = '\0';//Importante el '\0'
return res;
}

void ClaseCalculadora::botonMas_left_cb(){
operar('+');
}

void ClaseCalculadora::botonMenos_left_cb(){
operar('-');
}
```

```
void ClaseCalculadora::botonMul_left_cb(){
operar('*');
}

void ClaseCalculadora::botonDiv_left_cb(){
operar('/');
}

void ClaseCalculadora::botonIgual_left_cb(){
resolver();
}

void ClaseCalculadora::botonC_left_cb(){
inicializar();
}

void ClaseCalculadora::operar(char c){
op1 = atoi(output->value());
accion = 1;
operacion = c;
}

void ClaseCalculadora::resolver(){
char resultado[20];
op2 = atoi(output->value());
switch(operacion){
        case '+': itoa(op1 + op2, resultado, 10);
                        output->value(resultado);
                        break;
        case '-': itoa(op1 - op2, resultado, 10);
                        output->value(resultado);
                        break;
        case '*': itoa(op1 * op2, resultado, 10);
                        output->value(resultado);
```

```
                        break;
        case '/': if(op2 == 0){
                        output->value("ERROR");
                }
                else{
                        itoa(op1 / op2, resultado, 10);
                        output->value(resultado);
                }
                        break;
        default: output->value("ERROR");
}
}

void ClaseCalculadora::borrar(){
output->value("");
}

void ClaseCalculadora::inicializar(){
borrar();
accion = 0;
}

ClaseCalculadora::ClaseCalculadora(int x, int y, int w, int h):Fl_Window(x,y,w,h){
this->labelfont(0);
this->labelsize(14);
this->labelcolor(56);
this->align(1);
this->box((Fl_Boxtype)2);
this->color(49);
boton0 = new Fl_Button(100,400,100,100, "0");
this->add(boton0);
boton0->labelfont(0);
boton0->labelsize(50);
boton0->labelcolor(56);
```

```
boton0->align(0);
boton0->labeltype(FL_NORMAL_LABEL);
boton0->box((Fl_Boxtype)2);
boton0->color(93);
boton0->down_box((Fl_Boxtype)3);
boton0->selection_color(49);
boton0->callback(ClaseCalculadora_e::cb_boton0, this);
boton0->when(FL_WHEN_CHANGED);
boton1 = new Fl_Button(0,300,100,100, "1");
this->add(boton1);
boton1->labelfont(0);
boton1->labelsize(50);
boton1->labelcolor(56);
boton1->align(0);
boton1->labeltype(FL_NORMAL_LABEL);
boton1->box((Fl_Boxtype)2);
boton1->color(93);
boton1->down_box((Fl_Boxtype)3);
boton1->selection_color(49);
boton1->callback(ClaseCalculadora_e::cb_boton1, this);
boton1->when(FL_WHEN_CHANGED);
boton2 = new Fl_Button(100,300,100,100, "2");
this->add(boton2);
boton2->labelfont(0);
boton2->labelsize(50);
boton2->labelcolor(56);
boton2->align(0);
boton2->labeltype(FL_NORMAL_LABEL);
boton2->box((Fl_Boxtype)2);
boton2->color(93);
boton2->down_box((Fl_Boxtype)3);
boton2->selection_color(49);
boton2->callback(ClaseCalculadora_e::cb_boton2, this);
boton2->when(FL_WHEN_CHANGED);
```

```
boton3 = new Fl_Button(200,300,100,100, "3");
this->add(boton3);
boton3->labelfont(0);
boton3->labelsize(50);
boton3->labelcolor(56);
boton3->align(0);
boton3->labeltype(FL_NORMAL_LABEL);
boton3->box((Fl_Boxtype)2);
boton3->color(93);
boton3->down_box((Fl_Boxtype)3);
boton3->selection_color(49);
boton3->callback(ClaseCalculadora_e::cb_boton3, this);
boton3->when(FL_WHEN_CHANGED);
boton4 = new Fl_Button(0,200,100,100, "4");
this->add(boton4);
boton4->labelfont(0);
boton4->labelsize(50);
boton4->labelcolor(56);
boton4->align(0);
boton4->labeltype(FL_NORMAL_LABEL);
boton4->box((Fl_Boxtype)2);
boton4->color(93);
boton4->down_box((Fl_Boxtype)3);
boton4->selection_color(49);
boton4->callback(ClaseCalculadora_e::cb_boton4, this);
boton4->when(FL_WHEN_CHANGED);
boton5 = new Fl_Button(100,200,100,100, "5");
this->add(boton5);
boton5->labelfont(0);
boton5->labelsize(50);
boton5->labelcolor(56);
boton5->align(0);
boton5->labeltype(FL_NORMAL_LABEL);
boton5->box((Fl_Boxtype)2);
```

```
boton5->color(93);
boton5->down_box((Fl_Boxtype)3);
boton5->selection_color(49);
boton5->callback(ClaseCalculadora_e::cb_boton5, this);
boton5->when(FL_WHEN_CHANGED);
boton6 = new Fl_Button(200,200,100,100, "6");
this->add(boton6);
boton6->labelfont(0);
boton6->labelsize(50);
boton6->labelcolor(56);
boton6->align(0);
boton6->labeltype(FL_NORMAL_LABEL);
boton6->box((Fl_Boxtype)2);
boton6->color(93);
boton6->down_box((Fl_Boxtype)3);
boton6->selection_color(49);
boton6->callback(ClaseCalculadora_e::cb_boton6, this);
boton6->when(FL_WHEN_CHANGED);
boton7 = new Fl_Button(0,100,100,100, "7");
this->add(boton7);
boton7->labelfont(0);
boton7->labelsize(50);
boton7->labelcolor(56);
boton7->align(0);
boton7->labeltype(FL_NORMAL_LABEL);
boton7->box((Fl_Boxtype)2);
boton7->color(93);
boton7->down_box((Fl_Boxtype)3);
boton7->selection_color(49);
boton7->callback(ClaseCalculadora_e::cb_boton7, this);
boton7->when(FL_WHEN_CHANGED);
boton8 = new Fl_Button(100,100,100,100, "8");
this->add(boton8);
boton8->labelfont(0);
```

```
boton8->labelsize(50);
boton8->labelcolor(56);
boton8->align(0);
boton8->labeltype(FL_NORMAL_LABEL);
boton8->box((Fl_Boxtype)2);
boton8->color(93);
boton8->down_box((Fl_Boxtype)3);
boton8->selection_color(49);
boton8->callback(ClaseCalculadora_e::cb_boton8, this);
boton8->when(FL_WHEN_CHANGED);
boton9 = new Fl_Button(200,100,100,100, "9");
this->add(boton9);
boton9->labelfont(0);
boton9->labelsize(50);
boton9->labelcolor(56);
boton9->align(0);
boton9->labeltype(FL_NORMAL_LABEL);
boton9->box((Fl_Boxtype)2);
boton9->color(93);
boton9->down_box((Fl_Boxtype)3);
boton9->selection_color(49);
boton9->callback(ClaseCalculadora_e::cb_boton9, this);
boton9->when(FL_WHEN_CHANGED);
output = new Fl_Multiline_Output(0,0,400,100);
this->add(output);
output->labelfont(5);
output->labelsize(14);
output->labelcolor(56);
output->align(0);
output->labeltype(FL_NORMAL_LABEL);
output->box((Fl_Boxtype)2);
output->color(7);
output->textfont(5);
output->textsize(50);
```

```
output->textcolor(0);
botonMas = new Fl_Button(300,100,100,100, "+");
this->add(botonMas);
botonMas->labelfont(1);
botonMas->labelsize(50);
botonMas->labelcolor(56);
botonMas->align(0);
botonMas->labeltype(FL_NORMAL_LABEL);
botonMas->box((Fl_Boxtype)2);
botonMas->color(183);
botonMas->down_box((Fl_Boxtype)3);
botonMas->selection_color(49);
botonMas->callback(ClaseCalculadora_e::cb_botonMas, this);
botonMas->when(FL_WHEN_CHANGED);
botonMenos = new Fl_Button(300,200,100,100, "-");
this->add(botonMenos);
botonMenos->labelfont(1);
botonMenos->labelsize(50);
botonMenos->labelcolor(56);
botonMenos->align(0);
botonMenos->labeltype(FL_NORMAL_LABEL);
botonMenos->box((Fl_Boxtype)2);
botonMenos->color(183);
botonMenos->down_box((Fl_Boxtype)3);
botonMenos->selection_color(49);
botonMenos->callback(ClaseCalculadora_e::cb_botonMenos, this);
botonMenos->when(FL_WHEN_CHANGED);
botonMul = new Fl_Button(300,300,100,100, "*");
this->add(botonMul);
botonMul->labelfont(1);
botonMul->labelsize(50);
botonMul->labelcolor(56);
botonMul->align(0);
botonMul->labeltype(FL_NORMAL_LABEL);
```

```
botonMul->box((Fl_Boxtype)2);
botonMul->color(183);
botonMul->down_box((Fl_Boxtype)3);
botonMul->selection_color(49);
botonMul->callback(ClaseCalculadora_e::cb_botonMul, this);
botonMul->when(FL_WHEN_CHANGED);
botonDiv = new Fl_Button(300,400,100,100, "/");
this->add(botonDiv);
botonDiv->labelfont(1);
botonDiv->labelsize(50);
botonDiv->labelcolor(56);
botonDiv->align(0);
botonDiv->labeltype(FL_NORMAL_LABEL);
botonDiv->box((Fl_Boxtype)2);
botonDiv->color(183);
botonDiv->down_box((Fl_Boxtype)3);
botonDiv->selection_color(49);
botonDiv->callback(ClaseCalculadora_e::cb_botonDiv, this);
botonDiv->when(FL_WHEN_CHANGED);
botonIgual = new Fl_Button(200,400,100,100, "=");
this->add(botonIgual);
botonIgual->labelfont(5);
botonIgual->labelsize(50);
botonIgual->labelcolor(56);
botonIgual->align(0);
botonIgual->labeltype(FL_NORMAL_LABEL);
botonIgual->box((Fl_Boxtype)2);
botonIgual->color(79);
botonIgual->down_box((Fl_Boxtype)3);
botonIgual->selection_color(49);
botonIgual->callback(ClaseCalculadora_e::cb_botonIgual, this);
botonIgual->when(FL_WHEN_CHANGED);
botonC = new Fl_Button(0,400,100,100, "C");
this->add(botonC);
```

```
botonC->labelfont(5);
botonC->labelsize(50);
botonC->labelcolor(56);
botonC->align(0);
botonC->labeltype(FL_NORMAL_LABEL);
botonC->box((Fl_Boxtype)2);
botonC->color(79);
botonC->down_box((Fl_Boxtype)3);
botonC->selection_color(49);
botonC->tooltip("Borrar");
botonC->callback(ClaseCalculadora_e::cb_botonC, this);
botonC->when(FL_WHEN_CHANGED);

}

ClaseCalculadora::ClaseCalculadora(int x, int y, int w, int h, char*
label):Fl_Window(x,y,w,h,label){
this->labelfont(0);
this->labelsize(14);
this->labelcolor(56);
this->align(1);
this->box((Fl_Boxtype)2);
this->color(49);
boton0 = new Fl_Button(100,400,100,100, "0");
this->add(boton0);
boton0->labelfont(0);
boton0->labelsize(50);
boton0->labelcolor(56);
boton0->align(0);
boton0->labeltype(FL_NORMAL_LABEL);
boton0->box((Fl_Boxtype)2);
boton0->color(93);
boton0->down_box((Fl_Boxtype)3);
boton0->selection_color(49);
```

```
boton0->callback(ClaseCalculadora_e::cb_boton0, this);
boton0->when(FL_WHEN_CHANGED);
boton1 = new Fl_Button(0,300,100,100, "1");
this->add(boton1);
boton1->labelfont(0);
boton1->labelsize(50);
boton1->labelcolor(56);
boton1->align(0);
boton1->labeltype(FL_NORMAL_LABEL);
boton1->box((Fl_Boxtype)2);
boton1->color(93);
boton1->down_box((Fl_Boxtype)3);
boton1->selection_color(49);
boton1->callback(ClaseCalculadora_e::cb_boton1, this);
boton1->when(FL_WHEN_CHANGED);
boton2 = new Fl_Button(100,300,100,100, "2");
this->add(boton2);
boton2->labelfont(0);
boton2->labelsize(50);
boton2->labelcolor(56);
boton2->align(0);
boton2->labeltype(FL_NORMAL_LABEL);
boton2->box((Fl_Boxtype)2);
boton2->color(93);
boton2->down_box((Fl_Boxtype)3);
boton2->selection_color(49);
boton2->callback(ClaseCalculadora_e::cb_boton2, this);
boton2->when(FL_WHEN_CHANGED);
boton3 = new Fl_Button(200,300,100,100, "3");
this->add(boton3);
boton3->labelfont(0);
boton3->labelsize(50);
boton3->labelcolor(56);
boton3->align(0);
```

```
boton3->labeltype(FL_NORMAL_LABEL);
boton3->box((Fl_Boxtype)2);
boton3->color(93);
boton3->down_box((Fl_Boxtype)3);
boton3->selection_color(49);
boton3->callback(ClaseCalculadora_e::cb_boton3, this);
boton3->when(FL_WHEN_CHANGED);
boton4 = new Fl_Button(0,200,100,100, "4");
this->add(boton4);
boton4->labelfont(0);
boton4->labelsize(50);
boton4->labelcolor(56);
boton4->align(0);
boton4->labeltype(FL_NORMAL_LABEL);
boton4->box((Fl_Boxtype)2);
boton4->color(93);
boton4->down_box((Fl_Boxtype)3);
boton4->selection_color(49);
boton4->callback(ClaseCalculadora_e::cb_boton4, this);
boton4->when(FL_WHEN_CHANGED);
boton5 = new Fl_Button(100,200,100,100, "5");
this->add(boton5);
boton5->labelfont(0);
boton5->labelsize(50);
boton5->labelcolor(56);
boton5->align(0);
boton5->labeltype(FL_NORMAL_LABEL);
boton5->box((Fl_Boxtype)2);
boton5->color(93);
boton5->down_box((Fl_Boxtype)3);
boton5->selection_color(49);
boton5->callback(ClaseCalculadora_e::cb_boton5, this);
boton5->when(FL_WHEN_CHANGED);
boton6 = new Fl_Button(200,200,100,100, "6");
```

```
this->add(boton6);
boton6->labelfont(0);
boton6->labelsize(50);
boton6->labelcolor(56);
boton6->align(0);
boton6->labeltype(FL_NORMAL_LABEL);
boton6->box((Fl_Boxtype)2);
boton6->color(93);
boton6->down_box((Fl_Boxtype)3);
boton6->selection_color(49);
boton6->callback(ClaseCalculadora_e::cb_boton6, this);
boton6->when(FL_WHEN_CHANGED);
boton7 = new Fl_Button(0,100,100,100, "7");
this->add(boton7);
boton7->labelfont(0);
boton7->labelsize(50);
boton7->labelcolor(56);
boton7->align(0);
boton7->labeltype(FL_NORMAL_LABEL);
boton7->box((Fl_Boxtype)2);
boton7->color(93);
boton7->down_box((Fl_Boxtype)3);
boton7->selection_color(49);
boton7->callback(ClaseCalculadora_e::cb_boton7, this);
boton7->when(FL_WHEN_CHANGED);
boton8 = new Fl_Button(100,100,100,100, "8");
this->add(boton8);
boton8->labelfont(0);
boton8->labelsize(50);
boton8->labelcolor(56);
boton8->align(0);
boton8->labeltype(FL_NORMAL_LABEL);
boton8->box((Fl_Boxtype)2);
boton8->color(93);
```

```
boton8->down_box((Fl_Boxtype)3);
boton8->selection_color(49);
boton8->callback(ClaseCalculadora_e::cb_boton8, this);
boton8->when(FL_WHEN_CHANGED);
boton9 = new Fl_Button(200,100,100,100, "9");
this->add(boton9);
boton9->labelfont(0);
boton9->labelsize(50);
boton9->labelcolor(56);
boton9->align(0);
boton9->labeltype(FL_NORMAL_LABEL);
boton9->box((Fl_Boxtype)2);
boton9->color(93);
boton9->down_box((Fl_Boxtype)3);
boton9->selection_color(49);
boton9->callback(ClaseCalculadora_e::cb_boton9, this);
boton9->when(FL_WHEN_CHANGED);
output = new Fl_Multiline_Output(0,0,400,100);
this->add(output);
output->labelfont(5);
output->labelsize(14);
output->labelcolor(56);
output->align(0);
output->labeltype(FL_NORMAL_LABEL);
output->box((Fl_Boxtype)2);
output->color(7);
output->textfont(5);
output->textsize(50);
output->textcolor(0);
botonMas = new Fl_Button(300,100,100,100, "+");
this->add(botonMas);
botonMas->labelfont(1);
botonMas->labelsize(50);
botonMas->labelcolor(56);
```

```
botonMas->align(0);
botonMas->labeltype(FL_NORMAL_LABEL);
botonMas->box((Fl_Boxtype)2);
botonMas->color(183);
botonMas->down_box((Fl_Boxtype)3);
botonMas->selection_color(49);
botonMas->callback(ClaseCalculadora_e::cb_botonMas, this);
botonMas->when(FL_WHEN_CHANGED);
botonMenos = new Fl_Button(300,200,100,100, "-");
this->add(botonMenos);
botonMenos->labelfont(1);
botonMenos->labelsize(50);
botonMenos->labelcolor(56);
botonMenos->align(0);
botonMenos->labeltype(FL_NORMAL_LABEL);
botonMenos->box((Fl_Boxtype)2);
botonMenos->color(183);
botonMenos->down_box((Fl_Boxtype)3);
botonMenos->selection_color(49);
botonMenos->callback(ClaseCalculadora_e::cb_botonMenos, this);
botonMenos->when(FL_WHEN_CHANGED);
botonMul = new Fl_Button(300,300,100,100, "*");
this->add(botonMul);
botonMul->labelfont(1);
botonMul->labelsize(50);
botonMul->labelcolor(56);
botonMul->align(0);
botonMul->labeltype(FL_NORMAL_LABEL);
botonMul->box((Fl_Boxtype)2);
botonMul->color(183);
botonMul->down_box((Fl_Boxtype)3);
botonMul->selection_color(49);
botonMul->callback(ClaseCalculadora_e::cb_botonMul, this);
botonMul->when(FL_WHEN_CHANGED);
```

```
botonDiv = new Fl_Button(300,400,100,100, "/");
this->add(botonDiv);
botonDiv->labelfont(1);
botonDiv->labelsize(50);
botonDiv->labelcolor(56);
botonDiv->align(0);
botonDiv->labeltype(FL_NORMAL_LABEL);
botonDiv->box((Fl_Boxtype)2);
botonDiv->color(183);
botonDiv->down_box((Fl_Boxtype)3);
botonDiv->selection_color(49);
botonDiv->callback(ClaseCalculadora_e::cb_botonDiv, this);
botonDiv->when(FL_WHEN_CHANGED);
botonIgual = new Fl_Button(200,400,100,100, "=");
this->add(botonIgual);
botonIgual->labelfont(5);
botonIgual->labelsize(50);
botonIgual->labelcolor(56);
botonIgual->align(0);
botonIgual->labeltype(FL_NORMAL_LABEL);
botonIgual->box((Fl_Boxtype)2);
botonIgual->color(79);
botonIgual->down_box((Fl_Boxtype)3);
botonIgual->selection_color(49);
botonIgual->callback(ClaseCalculadora_e::cb_botonIgual, this);
botonIgual->when(FL_WHEN_CHANGED);
botonC = new Fl_Button(0,400,100,100, "C");
this->add(botonC);
botonC->labelfont(5);
botonC->labelsize(50);
botonC->labelcolor(56);
botonC->align(0);
botonC->labeltype(FL_NORMAL_LABEL);
botonC->box((Fl_Boxtype)2);
```

```
botonC->color(79);
botonC->down_box((Fl_Boxtype)3);
botonC->selection_color(49);
botonC->tooltip("Borrar");
botonC->callback(ClaseCalculadora_e::cb_botonC, this);
botonC->when(FL_WHEN_CHANGED);

}

ClaseCalculadora::ClaseCalculadora(int w, int h):Fl_Window(w,h){
this->labelfont(0);
this->labelsize(14);
this->labelcolor(56);
this->align(1);
this->box((Fl_Boxtype)2);
this->color(49);
boton0 = new Fl_Button(100,400,100,100, "0");
this->add(boton0);
boton0->labelfont(0);
boton0->labelsize(50);
boton0->labelcolor(56);
boton0->align(0);
boton0->labeltype(FL_NORMAL_LABEL);
boton0->box((Fl_Boxtype)2);
boton0->color(93);
boton0->down_box((Fl_Boxtype)3);
boton0->selection_color(49);
boton0->callback(ClaseCalculadora_e::cb_boton0, this);
boton0->when(FL_WHEN_CHANGED);
boton1 = new Fl_Button(0,300,100,100, "1");
this->add(boton1);
boton1->labelfont(0);
boton1->labelsize(50);
boton1->labelcolor(56);
```

```
boton1->align(0);
boton1->labeltype(FL_NORMAL_LABEL);
boton1->box((Fl_Boxtype)2);
boton1->color(93);
boton1->down_box((Fl_Boxtype)3);
boton1->selection_color(49);
boton1->callback(ClaseCalculadora_e::cb_boton1, this);
boton1->when(FL_WHEN_CHANGED);
boton2 = new Fl_Button(100,300,100,100, "2");
this->add(boton2);
boton2->labelfont(0);
boton2->labelsize(50);
boton2->labelcolor(56);
boton2->align(0);
boton2->labeltype(FL_NORMAL_LABEL);
boton2->box((Fl_Boxtype)2);
boton2->color(93);
boton2->down_box((Fl_Boxtype)3);
boton2->selection_color(49);
boton2->callback(ClaseCalculadora_e::cb_boton2, this);
boton2->when(FL_WHEN_CHANGED);
boton3 = new Fl_Button(200,300,100,100, "3");
this->add(boton3);
boton3->labelfont(0);
boton3->labelsize(50);
boton3->labelcolor(56);
boton3->align(0);
boton3->labeltype(FL_NORMAL_LABEL);
boton3->box((Fl_Boxtype)2);
boton3->color(93);
boton3->down_box((Fl_Boxtype)3);
boton3->selection_color(49);
boton3->callback(ClaseCalculadora_e::cb_boton3, this);
boton3->when(FL_WHEN_CHANGED);
```

```
boton4 = new Fl_Button(0,200,100,100, "4");
this->add(boton4);
boton4->labelfont(0);
boton4->labelsize(50);
boton4->labelcolor(56);
boton4->align(0);
boton4->labeltype(FL_NORMAL_LABEL);
boton4->box((Fl_Boxtype)2);
boton4->color(93);
boton4->down_box((Fl_Boxtype)3);
boton4->selection_color(49);
boton4->callback(ClaseCalculadora_e::cb_boton4, this);
boton4->when(FL_WHEN_CHANGED);
boton5 = new Fl_Button(100,200,100,100, "5");
this->add(boton5);
boton5->labelfont(0);
boton5->labelsize(50);
boton5->labelcolor(56);
boton5->align(0);
boton5->labeltype(FL_NORMAL_LABEL);
boton5->box((Fl_Boxtype)2);
boton5->color(93);
boton5->down_box((Fl_Boxtype)3);
boton5->selection_color(49);
boton5->callback(ClaseCalculadora_e::cb_boton5, this);
boton5->when(FL_WHEN_CHANGED);
boton6 = new Fl_Button(200,200,100,100, "6");
this->add(boton6);
boton6->labelfont(0);
boton6->labelsize(50);
boton6->labelcolor(56);
boton6->align(0);
boton6->labeltype(FL_NORMAL_LABEL);
boton6->box((Fl_Boxtype)2);
```

```
boton6->color(93);
boton6->down_box((Fl_Boxtype)3);
boton6->selection_color(49);
boton6->callback(ClaseCalculadora_e::cb_boton6, this);
boton6->when(FL_WHEN_CHANGED);
boton7 = new Fl_Button(0,100,100,100, "7");
this->add(boton7);
boton7->labelfont(0);
boton7->labelsize(50);
boton7->labelcolor(56);
boton7->align(0);
boton7->labeltype(FL_NORMAL_LABEL);
boton7->box((Fl_Boxtype)2);
boton7->color(93);
boton7->down_box((Fl_Boxtype)3);
boton7->selection_color(49);
boton7->callback(ClaseCalculadora_e::cb_boton7, this);
boton7->when(FL_WHEN_CHANGED);
boton8 = new Fl_Button(100,100,100,100, "8");
this->add(boton8);
boton8->labelfont(0);
boton8->labelsize(50);
boton8->labelcolor(56);
boton8->align(0);
boton8->labeltype(FL_NORMAL_LABEL);
boton8->box((Fl_Boxtype)2);
boton8->color(93);
boton8->down_box((Fl_Boxtype)3);
boton8->selection_color(49);
boton8->callback(ClaseCalculadora_e::cb_boton8, this);
boton8->when(FL_WHEN_CHANGED);
boton9 = new Fl_Button(200,100,100,100, "9");
this->add(boton9);
boton9->labelfont(0);
```

```
boton9->labelsize(50);
boton9->labelcolor(56);
boton9->align(0);
boton9->labeltype(FL_NORMAL_LABEL);
boton9->box((Fl_Boxtype)2);
boton9->color(93);
boton9->down_box((Fl_Boxtype)3);
boton9->selection_color(49);
boton9->callback(ClaseCalculadora_e::cb_boton9, this);
boton9->when(FL_WHEN_CHANGED);
output = new Fl_Multiline_Output(0,0,400,100);
this->add(output);
output->labelfont(5);
output->labelsize(14);
output->labelcolor(56);
output->align(0);
output->labeltype(FL_NORMAL_LABEL);
output->box((Fl_Boxtype)2);
output->color(7);
output->textfont(5);
output->textsize(50);
output->textcolor(0);
botonMas = new Fl_Button(300,100,100,100, "+");
this->add(botonMas);
botonMas->labelfont(1);
botonMas->labelsize(50);
botonMas->labelcolor(56);
botonMas->align(0);
botonMas->labeltype(FL_NORMAL_LABEL);
botonMas->box((Fl_Boxtype)2);
botonMas->color(183);
botonMas->down_box((Fl_Boxtype)3);
botonMas->selection_color(49);
botonMas->callback(ClaseCalculadora_e::cb_botonMas, this);
```

```
botonMas->when(FL_WHEN_CHANGED);
botonMenos = new Fl_Button(300,200,100,100, "-");
this->add(botonMenos);
botonMenos->labelfont(1);
botonMenos->labelsize(50);
botonMenos->labelcolor(56);
botonMenos->align(0);
botonMenos->labeltype(FL_NORMAL_LABEL);
botonMenos->box((Fl_Boxtype)2);
botonMenos->color(183);
botonMenos->down_box((Fl_Boxtype)3);
botonMenos->selection_color(49);
botonMenos->callback(ClaseCalculadora_e::cb_botonMenos, this);
botonMenos->when(FL_WHEN_CHANGED);
botonMul = new Fl_Button(300,300,100,100, "*");
this->add(botonMul);
botonMul->labelfont(1);
botonMul->labelsize(50);
botonMul->labelcolor(56);
botonMul->align(0);
botonMul->labeltype(FL_NORMAL_LABEL);
botonMul->box((Fl_Boxtype)2);
botonMul->color(183);
botonMul->down_box((Fl_Boxtype)3);
botonMul->selection_color(49);
botonMul->callback(ClaseCalculadora_e::cb_botonMul, this);
botonMul->when(FL_WHEN_CHANGED);
botonDiv = new Fl_Button(300,400,100,100, "/");
this->add(botonDiv);
botonDiv->labelfont(1);
botonDiv->labelsize(50);
botonDiv->labelcolor(56);
botonDiv->align(0);
botonDiv->labeltype(FL_NORMAL_LABEL);
```

```
botonDiv->box((Fl_Boxtype)2);
botonDiv->color(183);
botonDiv->down_box((Fl_Boxtype)3);
botonDiv->selection_color(49);
botonDiv->callback(ClaseCalculadora_e::cb_botonDiv, this);
botonDiv->when(FL_WHEN_CHANGED);
botonIgual = new Fl_Button(200,400,100,100, "=");
this->add(botonIgual);
botonIgual->labelfont(5);
botonIgual->labelsize(50);
botonIgual->labelcolor(56);
botonIgual->align(0);
botonIgual->labeltype(FL_NORMAL_LABEL);
botonIgual->box((Fl_Boxtype)2);
botonIgual->color(79);
botonIgual->down_box((Fl_Boxtype)3);
botonIgual->selection_color(49);
botonIgual->callback(ClaseCalculadora_e::cb_botonIgual, this);
botonIgual->when(FL_WHEN_CHANGED);
botonC = new Fl_Button(0,400,100,100, "C");
this->add(botonC);
botonC->labelfont(5);
botonC->labelsize(50);
botonC->labelcolor(56);
botonC->align(0);
botonC->labeltype(FL_NORMAL_LABEL);
botonC->box((Fl_Boxtype)2);
botonC->color(79);
botonC->down_box((Fl_Boxtype)3);
botonC->selection_color(49);
botonC->tooltip("Borrar");
botonC->callback(ClaseCalculadora_e::cb_botonC, this);
botonC->when(FL_WHEN_CHANGED);
```

```
}
```

Este fichero contiene casi toda la implementación. Al principio se agregan todas las declaraciones que el usuario introduzca dentro de la clase. Estas declaraciones son, en este caso, de variables globales que se van a necesitar para programar la funcionalidad de la calculadora. Después de esto aparecen las funciones asociadas a los callbacks y otras generadas a mano por el usuario, como la función “imprime”, “borrar”, etc.

Al final del fichero aparecen los tres constructores que se han añadido automáticamente por extender “ClaseCalculadora” a “Fl_Window”. Éstos se sacan de la lista de constructores, donde se guardan los argumentos de los distintos constructores de una clase y así se crean otros para “ClaseCalculadora” con los mismos argumentos que los de la superclase y que llaman a estos constructores de la superclase. El contendido de los tres constructores es el mismo y lo que hace es construir la interfaz gráfica de la calculadora creando uno a uno todos los atributos de la clase. Vamos a ver más detalladamente el caso de la creación del boton0 (tecla 0 de la calculadora), el código generado para crear ese botón es el siguiente:

```
boton0 = new Fl_Button(100,400,100,100, "0");
this->add(boton0);
boton0->labelfont(0);
boton0->labelsize(50);
boton0->labelcolor(56);
boton0->align(0);
boton0->labeltype(FL_NORMAL_LABEL);
boton0->box((Fl_Boxtype)2);
boton0->color(93);
boton0->down_box((Fl_Boxtype)3);
boton0->selection_color(49);
boton0->callback(ClaseCalculadora_e::cb_boton0, this);
boton0->when(FL_WHEN_CHANGED);
```

Primero se crea el botón llamando a su constructor con los argumentos que pone el usuario. Luego se añade el botón a this, que es la ventana de la calculadora. Las

siguientes nueve líneas fijan el estilo del botón. La siguiente línea es la que asocia al callback del botón la función cb_boton0, contenida en el fichero ClaseCalculadora_e.h. La línea que hay después del callback hace que el callback se ejecute cuando ocurra un cambio en el botón.

- Archivo "ClaseCalculadora_e.h"

```
// generated by Fast Light User Interface Designer (fluid)
#ifndef ClaseCalculadora_e_h
#define ClaseCalculadora_e_h

#include <FL/Fl.H>

class ClaseCalculadora_e{
public:
static void cb_boton0(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->boton0_left_cb();
}
}

static void cb_boton1(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->boton1_left_cb();
}
}

static void cb_boton2(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
```

```
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->boton2_left_cb();
}
}

static void cb_boton3(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->boton3_left_cb();
}
}

static void cb_boton4(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->boton4_left_cb();
}
}

static void cb_boton5(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->boton5_left_cb();
}
}

static void cb_boton6(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->boton6_left_cb();
```

```
}
}

static void cb_boton7(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->boton7_left_cb();
}
}

static void cb_boton8(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->boton8_left_cb();
}
}

static void cb_boton9(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->boton9_left_cb();
}
}

static void cb_botonMas(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->botonMas_left_cb();
}
}
```

```
static void cb_botonMenos(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->botonMenos_left_cb();
}
}

static void cb_botonMul(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->botonMul_left_cb();
}
}

static void cb_botonDiv(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->botonDiv_left_cb();
}
}

static void cb_botonIgual(Fl_Widget* o, void* v){
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->botonIgual_left_cb();
}
}

static void cb_botonC(Fl_Widget* o, void* v){
```

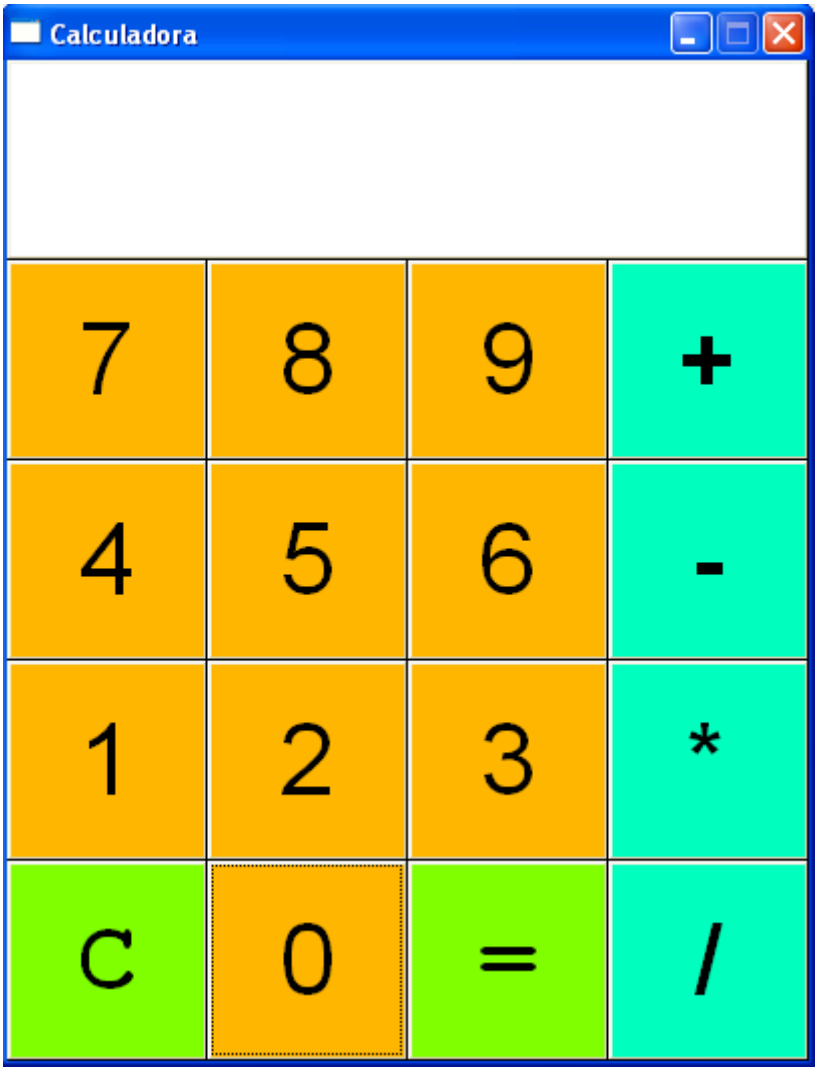
```
int evento=Fl::event();
int evento_boton=Fl::event_button();
if((evento==FL_RELEASE) && (evento_boton==1)){
((ClaseCalculadora*)v)->botonC_left_cb();
}
}


};

#endif
```

Este fichero consta de una clase que contiene una serie de funciones estáticas, que son las que se van a ejecutar cuando ocurra un evento en el widget correspondiente. En el caso de los botones por ejemplo, al principio de estas funciones se capturan en dos variables tanto el evento ocurrido como el botón del ratón que ha producido el evento, para poder distinguir distintos eventos asociados a un mismo objeto y ejecutar en cada caso el código correspondiente.

Hay dos casos de funciones asociadas a callbacks de widgets que no se corresponden con el caso explicado anteriormente. Si el widget es una clase que extiende a Fl_Window, puede tener otros dos eventos más (el FL_CLOSE y el FL_FOCUS), por lo que en este caso no habría que distinguir el botón de ratón que se pulsa en la condición de la función. El otro caso distinto es el de los Menu_Items y los Submenus, que solo tienen un evento, por lo que no existiría “if” para distinguir el tipo de evento y por tanto tampoco serían necesarias las variables que capturan el tipo de evento y el botón de ratón que ha desencadenado ese evento y por tanto no aparecerían.

Para que los archivos creados por este módulo puedan compilarse y ejecutarse hay que meterlos en un proyecto de Microsoft Visual C++, incluyendo en las propiedades del mismo las librerías necesarias para su compilación, que se corresponden con las que se han listado en el apartado de instalación de la librería FLTK en Windows. Una vez creado el proyecto e incluidas las librerías necesarias y los

ficheros de código C++ creados por la herramienta para el ejemplo de la calculadora, compilamos, ejecutamos y el resultado que se muestra es el siguiente:



Como podemos ver la ventana que se nos muestra es la misma que habíamos creado a mano con nuestro editor, y si ejecutamos cualquier acción vemos que la calculadora se comporta como una calculadora real, que hace sumas, restas, multiplicaciones y divisiones de números enteros.

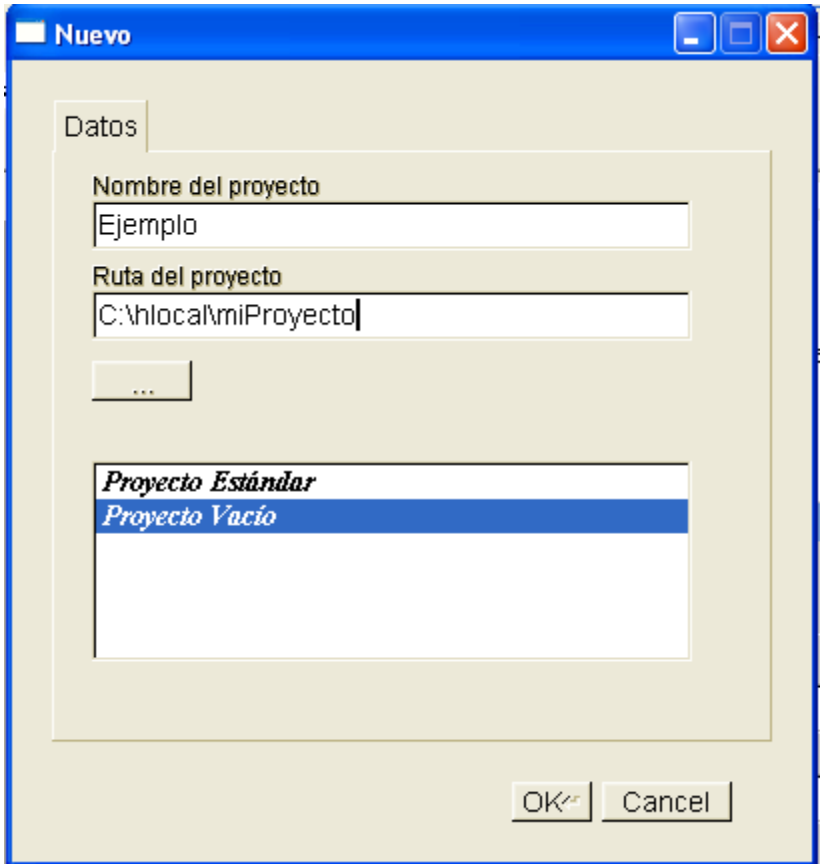
8. Manual de Usuario

Este es un sencillo tutorial para indicar al usuario los pasos a seguir para crear un proyecto.

- Requerimientos:
  - Windows: Mover a la carpeta System de Windows las dlls de Xerces y de FLTK.
  - Linux: instalar las librerías FLTK y Xerces y tener las librerías gráficas.

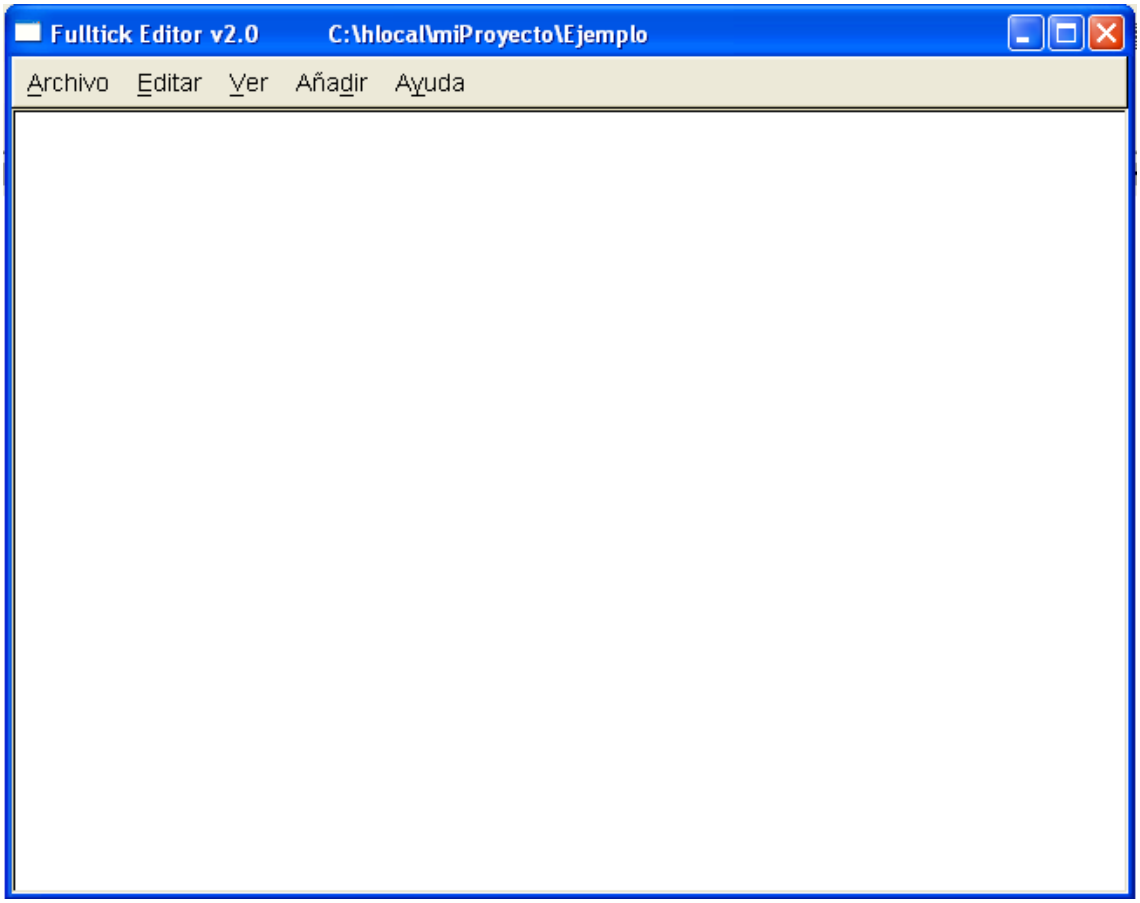
8.1 Proyecto Nuevo

Para ello seleccionamos en el menú "Archivo → Nuevo → Proyecto Vacío". A continuación escribimos el nombre del programa y la ruta donde lo queremos guardar (…).

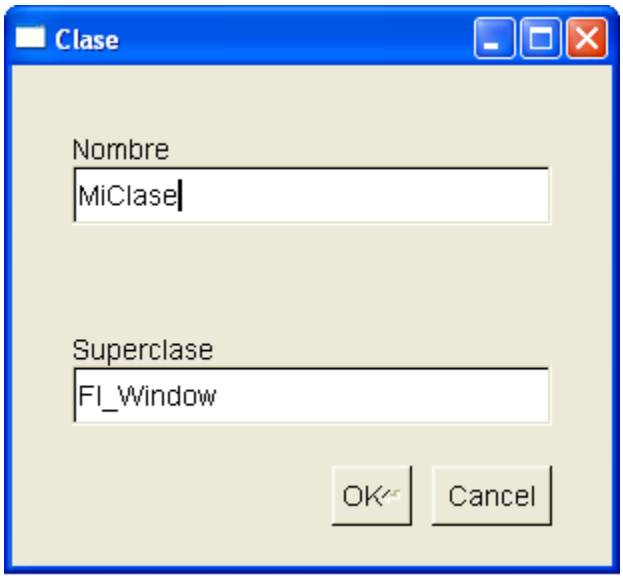


El proyecto estándar es un proyecto de prueba en el que el usuario puede ver el funcionamiento y las posibilidades de la aplicación.

En la parte superior del editor aparece la ruta donde se guardará nuestro programa. A continuación procedemos a crear nuestro programa.

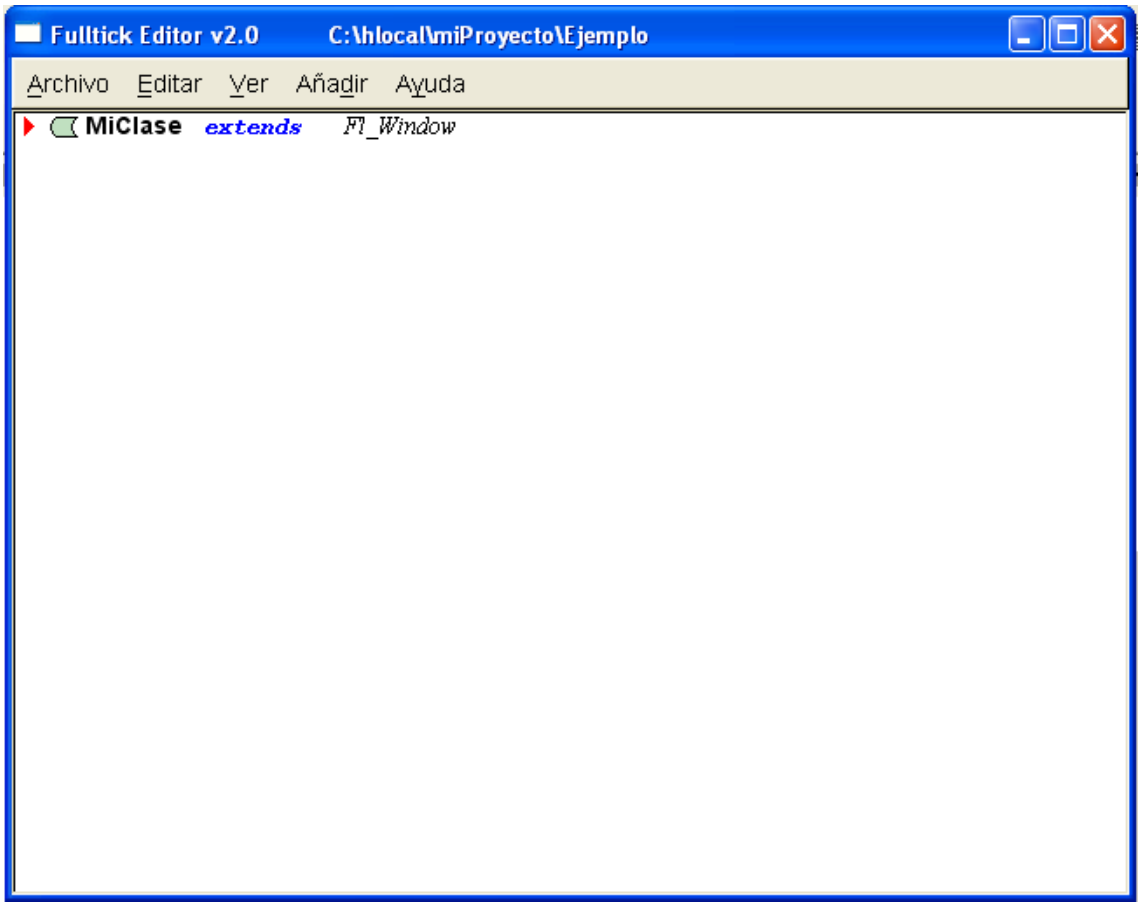


El único elemento que se puede agregar en un primer momento es una clase, por lo que en el menú seleccionamos “Añadir → Código → Clase”.



En “Nombre” ponemos el nombre de nuestra clase, y en “SuperClase” ponemos de qué superclase queremos que extienda nuestra clase (dentro de una clase no puede haber dos elementos con el mismo nombre, independientemente de que sean de tipos diferentes). Por defecto aparece Fl_Window, que es la superclase que deben tener las clases en las que queramos dibujar elementos gráficos. Al ser su padre Fl_Window, nos aparecerá por defecto una ventana.

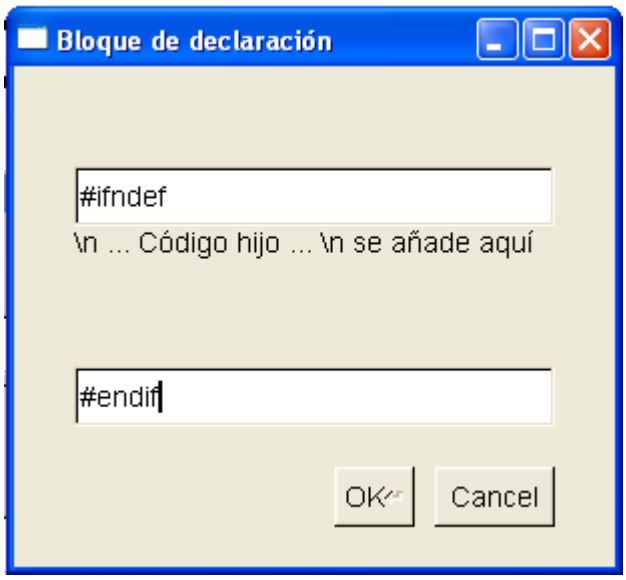
En el editor nos aparece esto:



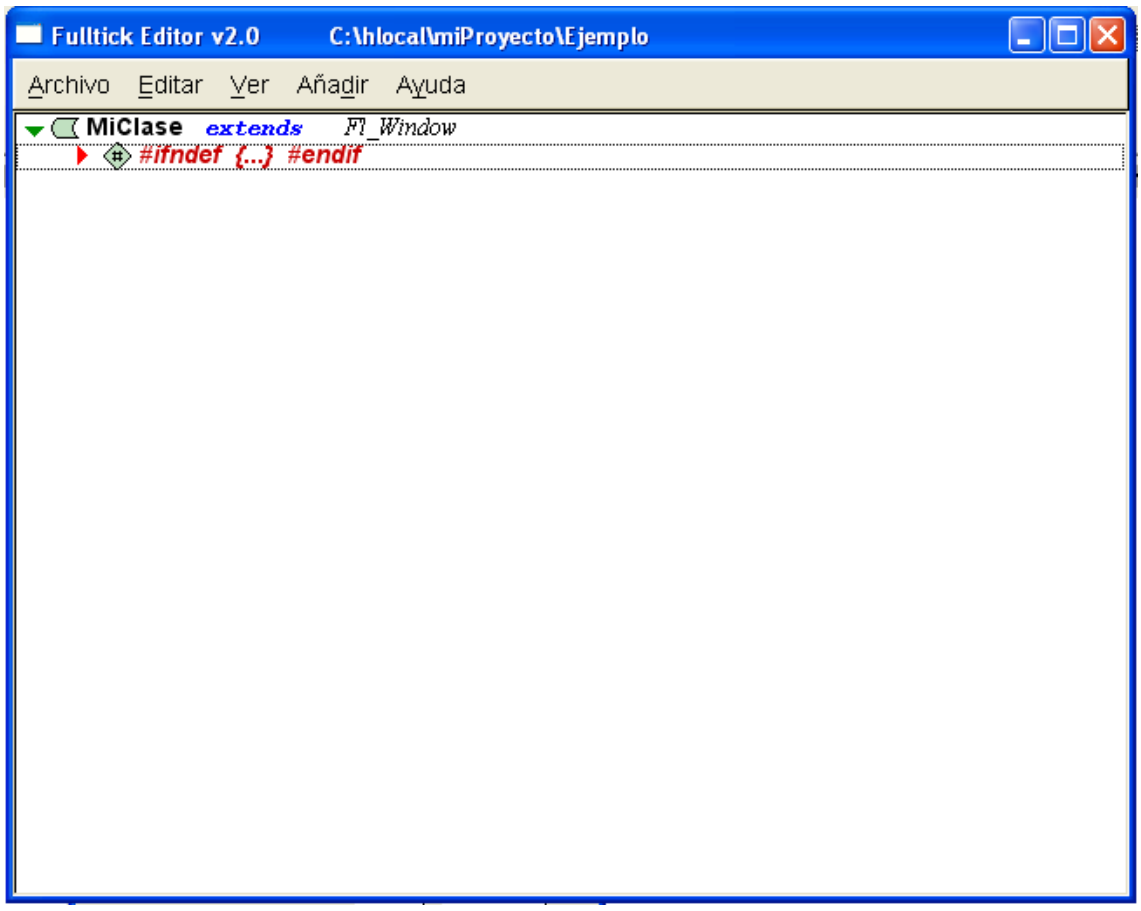
Una vez creada la clase le podemos añadir cualquier otro elemento (función, declaración, código, bloque de declaraciones, bloque de código). Hay ciertas restricciones a la hora de añadir elementos, que son las siguientes:

- Las clases sólo se pueden añadir a la raíz del proyecto.
- Las funciones sólo se pueden añadir dentro de una clase.
- Las declaraciones se pueden añadir dentro de una clase, una función o de los bloques de código ó declaraciones.
- El código se puede añadir dentro de una función o a un bloque de código.
- Los bloques declaraciones, se pueden añadir dentro de los bloques de declaraciones o de código, dentro de una función o de una clase.
- Los bloques de código pueden añadirse dentro de una función y dentro de bloques de código.

Los bloques tanto de código como de declaración, son expresiones de tipo condición, y en los dos campos de texto se deben añadir el comienzo y el fin de la condición, como por ejemplo:

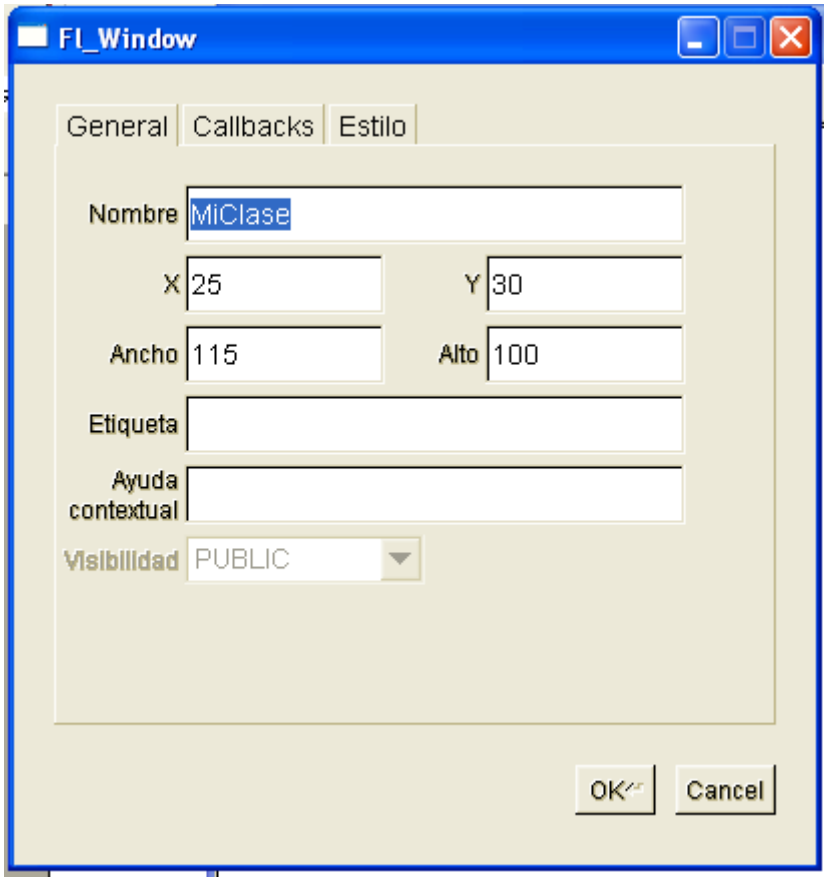


Y en el editor tenemos:



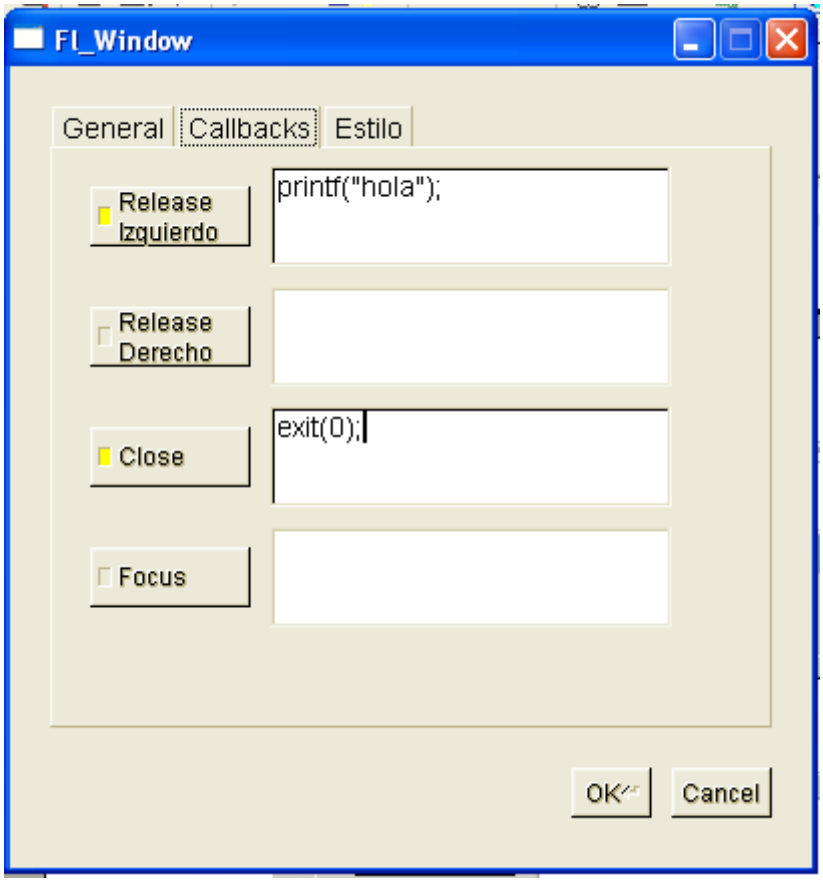
Al hacer doble clic sobre cualquier elemento situado en el editor, nos aparecerá una ventana con sus datos, los cuales podrán ser modificados por el usuario.

Para poder añadir otros elementos como botones, campos de texto…, nuestra clase debe tener como padre a Fl_Window.



Al crear la ventana nos aparecen tres pestañas, una es la principal, la otra está dedicada a las acciones que queremos que se realicen cuando se produzca un evento del ratón y la otra es la dedicada a los estilos.

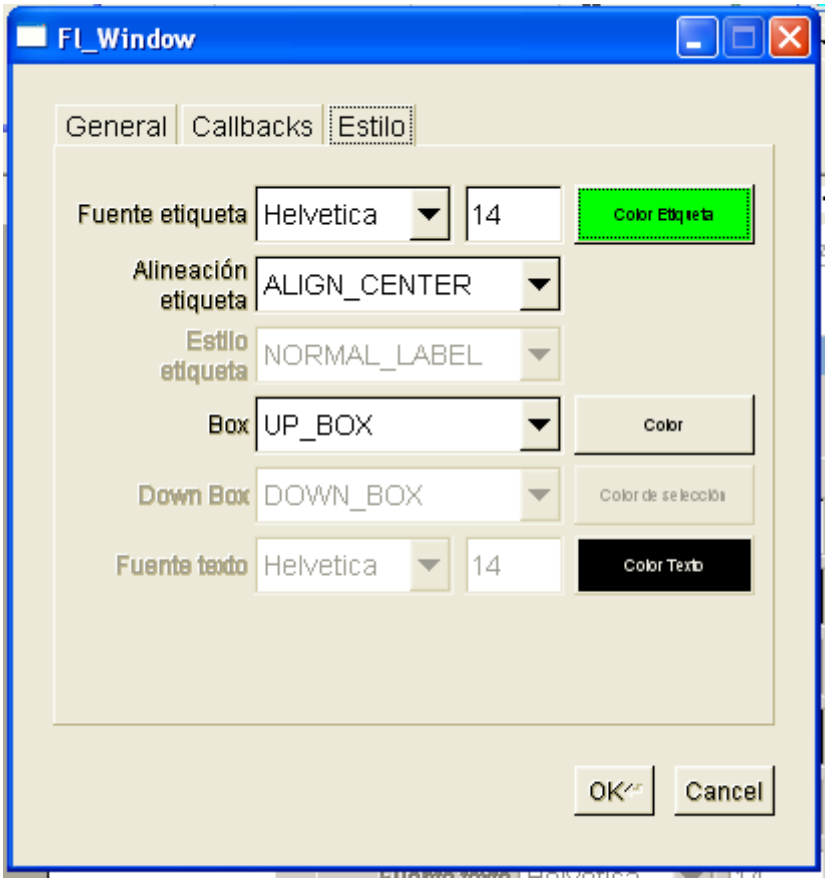
Pestaña para los eventos del ratón:



En este caso queremos que al dejar de pulsar el botón izquierdo del ratón sobre la ventana, nos escribirá “Hola” en la consola de comandos. Si cerramos la ventana, se produce el evento close y se realiza la acción exit(0). Para poder realizar un evento primero debemos seleccionar el correspondiente evento y marcarlo (aparecen en amarillo en la figura superior).

Las clases que extienden de Fl_Window son los únicos elementos que disponen de los eventos close y focus, mientras que el resto de elementos sólo constan de eventos para release de los botones del ratón (las ventanas también tienen estos eventos), excepto en el caso de los Menu_Item y los Submenu que solo tienen release con el botón izquierdo.

La pestaña de estilo nos permite elegir los tipos de letra, colores, tipo de fuente, etc. que queramos que tenga nuestro elemento.

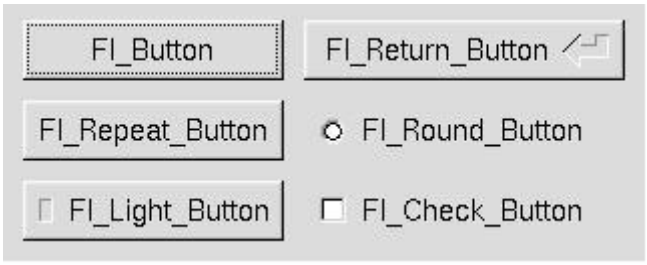


En esta figura se ven algunos campos inhabilitados, esto es porque no todos los elementos gráficos permiten las mismas opciones.

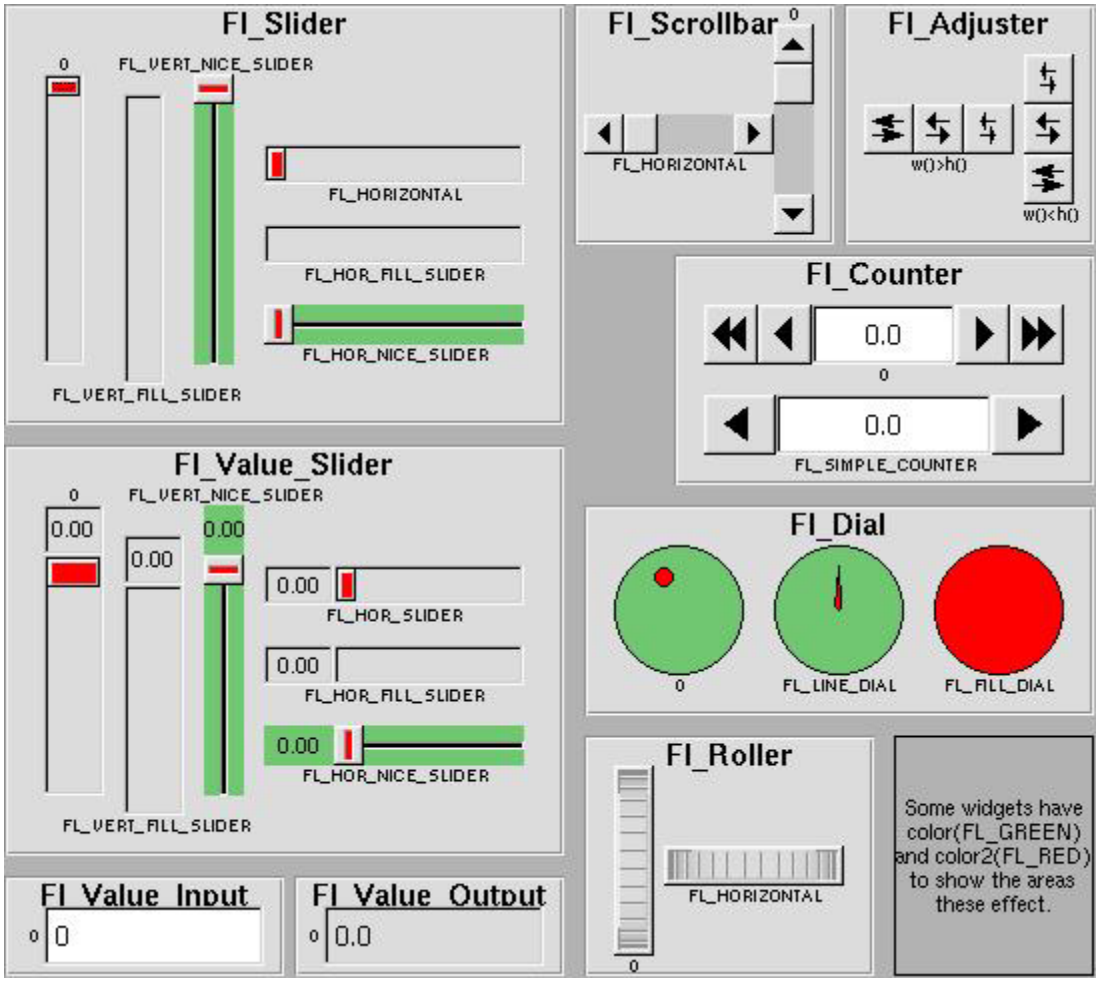
Con nuestra ventana creada, ahora podemos añadirle botones, valuators, texto, menú, explorador y otros. Para añadirlos, primero seleccionar la ventana y luego en el menú seleccionar “Añadir” y el elemento correspondiente. Al añadir este elemento nos aparecerá una ventana de diseño como la que apareció al crear la ventana, para establecer las dimensiones del elemento y las acciones que queremos que se realicen al producirse eventos del ratón.

Algunos ejemplos de elementos que podemos añadir a una ventana son:

Botones:

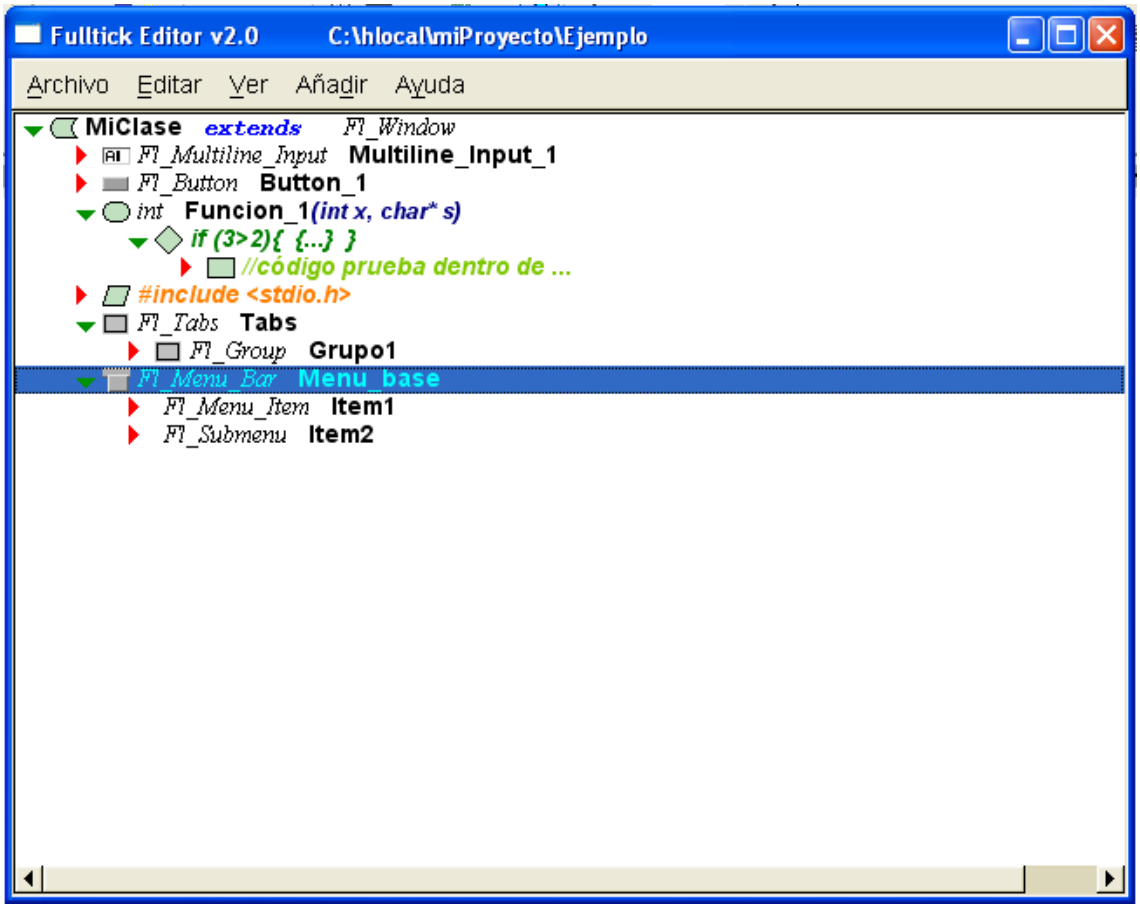


Valuators:



Con esto ya disponemos de las nociones básicas para crear nuestra interfaz gráfica.

Ahora mostramos un ejemplo de como quedaría un proyecto:



En primer lugar se ha creado una clase "Class_1" cuyo padre es "Fl_Window". Los elementos se muestran de esta forma "*Tipo* **NombreElemento**", donde Tipo indica que tipo de elemento es (campo de texto, botón…) y NombreElemento es el nombre que el usuario ha dado a ese elemento.

Dentro de nuestra clase hemos creado un campo de texto y un botón.

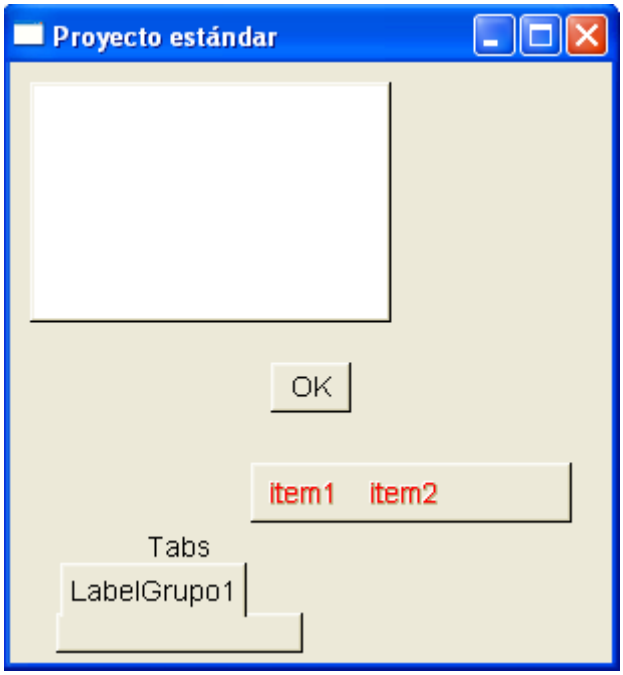
Posteriormente tenemos "Funcion_1" que es una función que tiene como parámetros (int x,char* s) y devuelve un int. Si la clase se hubiese llamado igual que la función, esta se interpreta como que es un constructor de la clase. Dentro de esta función hemos añadido un bloque de código "if (3>2){ {…} }". Todos los bloques de código tendrán un color verde oscuro. El cuerpo de este bloque de código es "//código prueba dentro de…". El código aparecerá en nuestro editor en un color verde claro. También tenemos un bloque de declaraciones "#ifndef {…} #endif", los cuales siempre aparecerán en rojo.

A continuación se encuentra una declaración “#include <stdio.h>”. Todas las declaraciones aparecerán en el editor en color naranja.

Por último en este ejemplo aparece una pestaña con un grupo y luego un menú en el que se pueden añadir elementos de forma que quede como un desplegable.

Como se puede ver en la parte superior del editor, aparecen tanto la versión de la aplicación como la ruta en la que estamos guardando nuestro proyecto.

Visualmente este proyecto quedaría de la siguiente forma:



8.2 Guardar Proyecto

El siguiente paso consiste en generar unos ficheros intermedios (XML), de los cuales el usuario puede despreocuparse y que sirven para posteriormente generar el código C++ de nuestra interfaz. Para ello en el menú seleccionamos “Archivo →

Guardar" y se nos crearán un fichero XML por cada clase que tenga nuestro editor, así como un documento de texto y un fichero XML general para el proyecto .Estos archivos se crean dentro de la carpeta XML.

Los XML siguen una notación a la hora de crearse, de forma que se llamarán "NombreClase.xml" y "NombreProyecto.xml". El archivo de texto se llamará "NombreProyecto.fl".

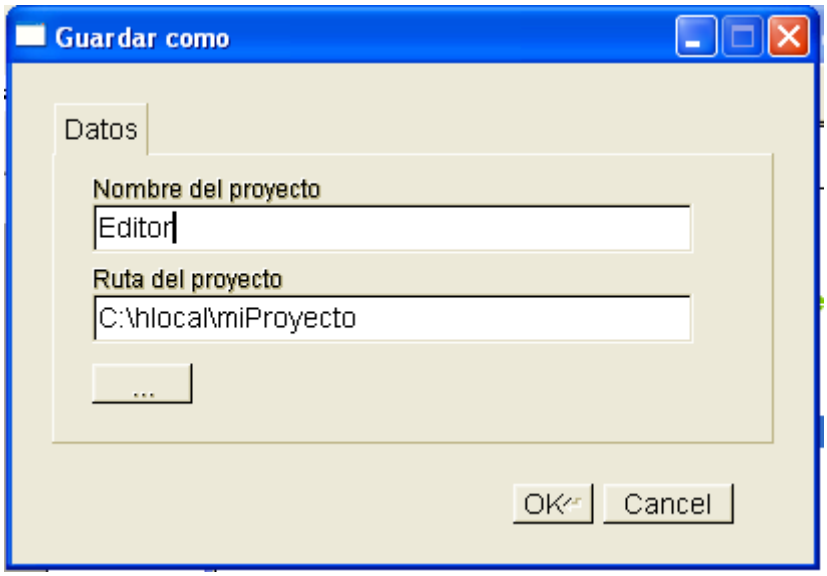
El archivo "NombreClase.xml" sirve para poder generar el código C++ del ".h" y ".cxx" de esa clase y el archivo "NombreProyecto.xml" sirve para poder generar el código del ".cxx" y del ".h" de la clase que tendrá la función main de nuestro proyecto.

El "NombreProyecto.fl" contiene los nombres de todos los ficheros XML que se hayan guardado en nuestro proyecto.

Cada vez que se haga "Archivo → Guardar" se vuelven a crear los ficheros XML, por lo que se borrará el contenido anterior que estos archivos tuvieran.

8.3 Guardar Proyecto Como

Esta es una forma alternativa de guardar un proyecto. Si el usuario selecciona "Archivo → Guardar Como" le aparecerá una ventana en la que el usuario elegirá la ruta en la que quiere guardar el proyecto. Los archivos que se crean son los mismos que en la opción "Guardar Proyecto".



8.4 Abrir Proyecto

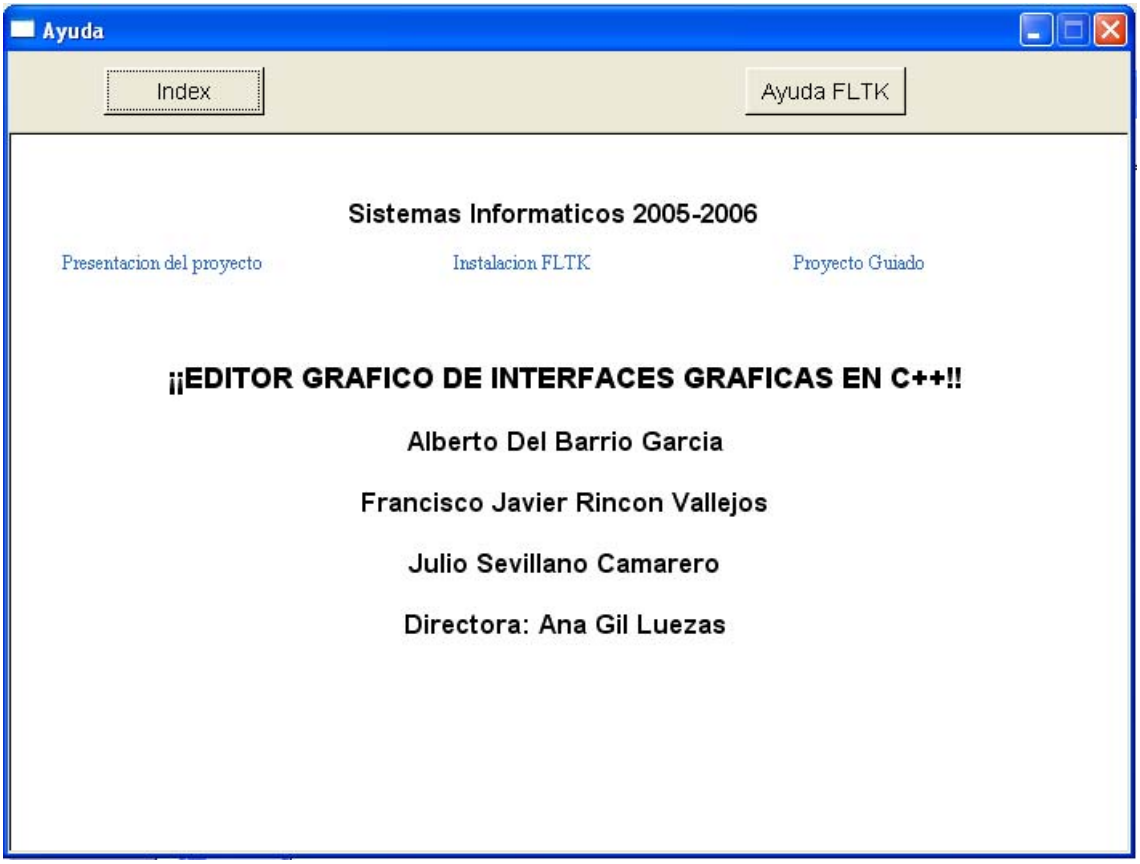
Para abrir un proyecto nuevo debemos seleccionar "Archivo → Abrir". A continuación nos aparecerá un explorador que nos permitirá elegir el proyecto que queremos abrir (se debe elegir el fichero "NombreProyecto.fl"). Hecho esto en la interfaz del editor se mostrará el proyecto previamente guardado.

8.5 Generar Código

Cuando se crean los archivos XML estamos en disposición de crear los ficheros .cxx y .h de nuestra interfaz. Se genera un .h y un .cxx por cada clase con el mismo nombre del XML y un .h y un .cxx para el proyecto, que contiene el main donde se crean todas las clases del proyecto. Si la clase contiene eventos (ya sea en la ventana principal o en alguno de sus atributos) se crea además un fichero .h llamado "NombreClase_e.h" que tiene una clase llamada "NombreClase_e" con todos las funciones asociadas a los eventos.

8.6 Ayuda

En el caso de que el usuario se encuentre desorientado existen unas secciones de ayuda para resolver posibles dudas, tanto sobre la librería FLTK como sobre esta aplicación.



Si pulsa en el menú "Ayuda → Sobre" le aparecerá esta pantalla. El botón "Index" lleva a esta misma página de inicio cuando estemos navegando por la ayuda y el botón "Ayuda FLTK" nos lleva a la documentación oficial de la librería FLTK.

En cuanto a la ayuda relacionada con nuestra aplicación destacar el apartado "Proyecto Guiado" que es un breve tutorial sobre como hacer un proyecto con esta aplicación.

Si pulsa en el menú "Ayuda → Manual" le llevará a la documentación oficial de FLTK, a la cual también se puede acceder de la forma anteriormente explicada.

Nótese que la carpeta de *Ayuda* debe estar en el mismo directorio que el ejecutable del editor.

9. Conclusiones y Líneas futuras de trabajo

Con este proyecto hemos conseguido desarrollar una aplicación que nos permita generar interfaces gráficas y un código asociado a la misma lo más inteligible y orientado a objetos posible.

Esta herramienta proporciona un ahorro de esfuerzo a la hora de desarrollar aplicaciones porque el código generado está libre de errores. Es especialmente útil en el desarrollo de prototipos.

Además nuestra aplicación tiene un uso claramente didáctico en la programación orientada a objetos.

Al tener un diseño modular, si en el futuro apareciera una librería FLTK para Java u otro lenguaje de programación se podría generar código para este lenguaje cambiando únicamente una parte del módulo 3.

El desarrollo de esta aplicación nos ha permitido aplicar conocimientos adquiridos durante la carrera (Ingeniería del Software, laboratorios de programación, Programación Orientada a Objetos, etc) y los adquiridos por nuestra propia experiencia, así como el uso de diferentes herramientas. Algunos conocimientos y herramientas utilizados son:

- Programación orientada a objetos con C++
- XML
- HTML
- FLTK
- Xerces
- Microsoft Visual C++ 6.0
- Eclipse
- StarUML
- Windows XP
- Linux Debian

Así mismo, tal y como está diseñada la aplicación permitiría realizar ciertas modificaciones o ampliaciones que los desarrolladores de este proyecto u otros en un futuro quisieran realizar, sin que suponga mucho trabajo para los mismos.

Entre las mejoras que se podrían realizar sugerimos:

- Aumentar las funcionalidades de edición, ya que ahora mismo sólo se permiten Borrar y Seleccionar Todo.
- Aumentar el número de elementos que se pueden añadir a la interfaz.
- Permitir crear nuestras propias "plantillas de clase", que el usuario pueda reutilizar en un futuro.
- Desarrollar algún tipo de preprocesador que permita al usuario conocer si la estructura de datos que se está construyendo sobre el editor producirá errores de compilación, pero antes de compilar.
- Ahora mismo el usuario puede introducir como nombre de los elementos lo que quiera, por lo que se podrían establecer restricciones de forma que los nombres de los elementos sean identificadores de C++.
- Sería interesante integrarlo en un compilador del tipo Eclipse o similares, de forma que con nuestra aplicación se pudieran crear las interfaces gráficas.
- Hacer un instalador que incluya las dlls y las librerías para que el usuario no tenga que preocuparse de los pormenores de la configuración de la aplicación.
- Distinguir más tipos de superclases, ya que ahora mismo sólo se puede extender a Fl_Window.
- Se debería añadir en el editor la posibilidad de añadir atributos privados en las clases, que sería simplemente añadirlo en la parte de añadir código como un tipo diferente a los ahí incluidos y modificar los módulos 2 y 3 para que traten este nuevo elemento.

Bibliografía

Para el desarrollo de este proyecto nos hemos basado en la información obtenida de estos sitios entre otros:

[1] Página oficial de FLTK, desde donde se puede descargar la librería y su manual.
http://www.fltk.org

[2] Tutorial sobre C++
http://c.conclase.net/

[3] Página oficial de la herramienta StarUML con la que hemos hecho los diagramas UML.
http://www.staruml.com/

[4] Página oficial de Xerces desde la que se puede consultar la ayuda y descargar la librería.
http://xerces.apache.org/xerces-j/

[5] Foro donde se comentan ciertas librerías de C++.
http://discuss.fogcreek.com/joelonsoftware1/default.asp?cmd=show&ixPost=20844&ixReplies=9

[6] Instrucciones y comandos C++.
http://www.lawebdelprogramador.com/temas/tema_manual_c.php

[7] Información sobre FOX
http://www.fox-toolkit.org/

[8] Tutorial sobre Linux.
http://usuarios.lycos.es/vinosec/Linux/Linux.html

[9] Información sobre FLTK.
http://freshmeat.net/projects/fltk/

[10] Página con diferentes enlaces a librerías de C++ (FLTK,GTK,Qt) entre otras cosas http://www.tldp.org/linuxfocus/English/October2004/article350.shtml

[11] XML. Rusty Harold, Elliote; Means, W. Scott .Editorial:Anaya Multimedia.

[12] XML Bible 2nd Edition. Elliote Rusty Harold.

Autorización

Los autores de este proyecto autorizan a la Universidad Complutense de Madrid a difundir y a utilizar con fines académicos no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación, y/o el prototipo desarrollado.

Alberto Antonio Del Barrio García

Francisco Javier Rincón Vallejos

Julio Sevillano Camarero