Code::Blocks (autore: Vittorio Albertoni)

Premessa

Code::Blocks è un software open source sulla cui origine e su chi sia il principale ideatore non si trovano notizie.

La prima release stabile risale al 2008.

Lo troviamo all'indirizzo

https://www.codeblocks.org/

Viene presentato come un editor per i linguaggi C e C++ estensibile attraverso plugin.

Aprendo la finestra di benvenuto ci viene presentata una panoramica secondo la quale con Code::Blocks potremmo fare di tutto.

La realtà, almeno per i dilettanti cui dedico i miei manualetti, è che Code::Blocks è un ottimo editor per programmare in C e C++ e, con poco sforzo, almeno per chi lavora su Linux, diventa un facile strumento per vestire di GUI i nostri programmi in C++.

Se ci fermiamo a questo non mi risulta si possa trovare di meglio.

Per fare altro le cose si complicano, ci portano al di fuori dalle capacità di un dilettante e molto spesso non ne vale la pena in quanto troviamo altri software con cui le stesse cose si fanno meglio e con meno fatica.

In questo manualetto descrivo il funzionamento di Code::Blocks per programmare in C e C++, dando per scontato che il lettore conosca questi linguaggi, per i quali esiste un'ampia disponibilità di manuali e guide, cartacee o in linea, anche in lingua italiana e alla portata di principianti.

Dal momento che non si tratta di linguaggi facili, ben venga Code::Blocks a renderli più accessibili.

Indice

1 Installazione di base **3**

2 Programmi console **3**

3 Installazione dei plugin per la grafica **4**

4 Impostazione del progetto per programma con GUI **6**

5 Costruzione della GUI **10**
5.1 Layout 11
5.2 Widget indispensabili per una GUI 11
5.2.1 wxStaticText 12
5.2.2 wxTextCtrl 12
5.2.3 wxButton 13
5.2.4 wxPanel 13

6 Alcuni esempi **13**

7 Conclusioni **20**

1 Installazione di base

All'indirizzo *https://www.codeblocks.org/* citato in Premessa, se apriamo la pagina DOWNLOADS troviamo quanto serve per installare Code::Blocks.

Nel momento in cui scrivo (gennaio 2024) l'ultima release è la 20.3, del marzo 2020.

Linux

Deve preventivamente essere disponibile sul sistema il compilatore GCC e devono essere disponibili i comandi `gcc` e `g++`.

Per chi usa Linux la via più facile è quella di installare Code::Blocks utilizzando il gestore dei pacchetti (Synaptic, Yum, Pamac, ecc, a seconda della distro) scegliendo la voce `codeblocks`. Ciò installa la versione di base del software, per programmare senza grafica, con le necessarie dipendenze. Verificare che venga installato anche `xterm` ed eventualmente installarlo a parte.

In alternativa occorre scaricare i file binari e installare `codeblocks`, `codeblocks-common` e `libcodeblock0`, installando a parte `xterm`. Ma è questa una strada tortuosa e sconsigliabile.

Windows

Se sul computer è già disponibile il compilatore MinGW (l'equivalente di GCC per Windows) si scarica e si utilizza il file `codeblocks-20.03-setup.exe`.

Altrimenti si può scaricare ed utilizzare il file `codeblocks-20.03mingw-setup.exe`, che installa anche il compilatore MinGW.

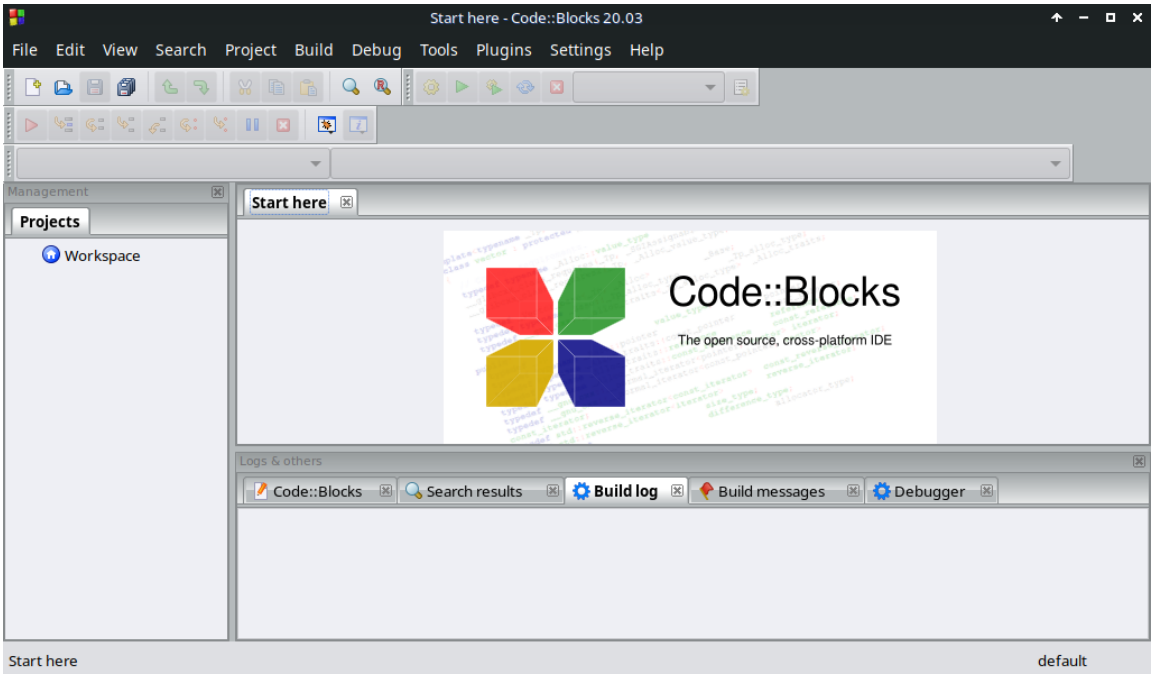
Nell'installazione è bene tenere selezionate tutte le opzioni proposte, compresa `Contrib plugin`.

Mac

Al momento la versione 20.3 non è disponibile per Mac e si può ovviare utilizzando il file `CodeBlocks-13.12-mac.zip`.

2 Programmi console

Se lanciamo Code::Blocks dopo l'installazione di base abbiamo la seguente finestra



Dal momento che la finestra START HERE, oltre che inutile può essere fuorviante, per quanto mi riguarda, eviterei che si ripresentasse.
Ciò si ottiene da menu SETTINGS ▷ ENVIRONMENT, scegliendo la scheda VIEW e deselezionando la voce SHOW "START HERE".

Da menu FILE ▷ NEW scegliamo cosa vogliamo fare e procediamo utilizzando Code::Blocks come un semplice editor, tuttavia arricchito da una formidabile code completion per i linguaggi C e C++ che ci facilita di molto la programmazione con questi linguaggi.

Scritto il programma, avendolo opportunamente memorizzato in un file con estensione `.c` o `.cpp` a seconda del linguaggio utilizzato, lo si compila da menu BUILD ▷ BUILD o premendo il pulsante nella barra degli strumenti e lo si può poi eseguire da menu BUILD ▷ RUN o premendo il pulsante nella barra degli strumenti.

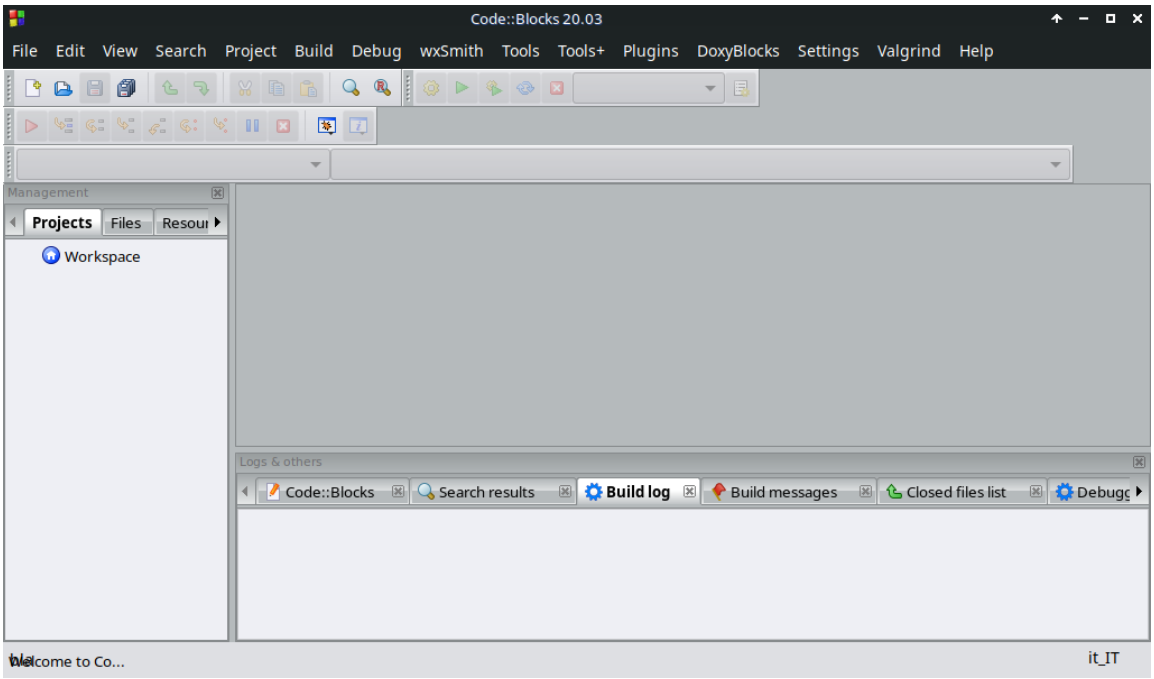
3 Installazione dei plugin per la grafica

Il primo plugin, molto utile, è un designer visuale di interfaccia utente grafica (GUI) che si chiama **wxSmith**.

Se abbiamo installato Code::Blocks su Windows utilizzando uno dei due installer indicati nel Capitolo 1, selezionando tra le opzioni di installazione `Contrib plugin` abbiamo già installato anche wxSmith.

Su Linux lo facciamo installando il pacchetto `codeblocks-contrib` con un gestore di pacchetti, che si occupa di installare anche le dipendenze necessarie.

Se tutto ha funzionato e se abbiamo fatto in modo che non si presenti più la finestra START HERE il nostro Code::Blocks arricchito si presenta così



E' quello della pagina precedente, senza la finestra START HERE e con l'aggiunta della voce WXSMITH nella barra del menu.

Dal momento che il designer wxSmith lavora con la libreria **wxWidgets**, ora dobbiamo installare questa libreria.
Si tratta di una libreria open source di componenti elementari per costruire un'interfaccia grafica (GUI).

L'installazione di queste librerie, almeno per chi non lavora su Linux, non è una cosa banale.

Linux

Qui la vita è facile.

Dobbiamo semplicemente installare con il gestore dei pacchetti della nostra distro (Synaptic, Yum, Pamac, ecc.) questi pacchetti:

`libwxbase3.0-0v5`
`libwxbase3.0-dev`
`libwxgtk3.0-gtk3-0v5`
`libwxgtk3.0-gtk3-dev`

Il primo potrebbe essere già installato.

Le versioni proposte sono tutte 3.0 in quanto al momento in cui scrivo e nel repository del computer da cui scrivo trovo questa versione per tutti i pacchetti. In momenti diversi e in repository diverse possono essere presenti altre versioni: l'importante è che tutti i pacchetti siano della stessa versione (tutti 3.0, tutti 3.1, tutti 3.2.1, ecc.).

Windows

Se lavoriamo sul sistema operativo Windows dobbiamo innanzi tutto andare all'indirizzo
https://www.wxwidgets.org/downloads/
e scaricare il file installer
`wxMSW-xx-Setup.exe`
dove `xx` indica la versione proposta.

Eseguendo questo file installiamo le wxWidgets nella directory C:\wxWidgets-xx.

Ora dobbiamo procedere alla loro compilazione con questa procedura:

. apriamo il prompt dei comandi con `cmd` e andiamo nella directory per la compilazione con il comando

```
cd \wxWidgets-xx\build\msw
```

. posizionati in questa directory scriviamo, su una sola riga, il seguente comando per la compilazione

```
mingw32-make -f makefile.gcc USE_XRC=1 SHARED=1 MONOLITHIC=1 ...
...  BUILD=release UNICODE=1 USE_OPENGL=1 VENDOR=cb ...
...  CXXFLAGS="-fno-keep-inline-dllexport"
```

La compilazione porterà via alcune decine di minuti: dipende dalla velocità della macchina su cui stiamo lavorando.

I numeri `1` associati alle varie opzioni ne denotano l'attivazione (altrimenti il numero darebbe `0`).
Per ciò che dovremo fare successivamente, ricordiamo di aver attivato l'opzione `MONOLITHIC` e `UNICODE` e che, per l'opzione `BUILD` abbiamo scelto `release` e non `debug`.

Mac

Anche su Mac dovremmo procedere alla installazione ed alla compilazione delle wxWidgets con Cocoa, ma fino a quando non sarà nuovamente disponibile per Mac un rilascio di Code::Blocks più recente di quello indicato nel Capitolo 1 è inutile perdere tempo per questa cosa.

* * *

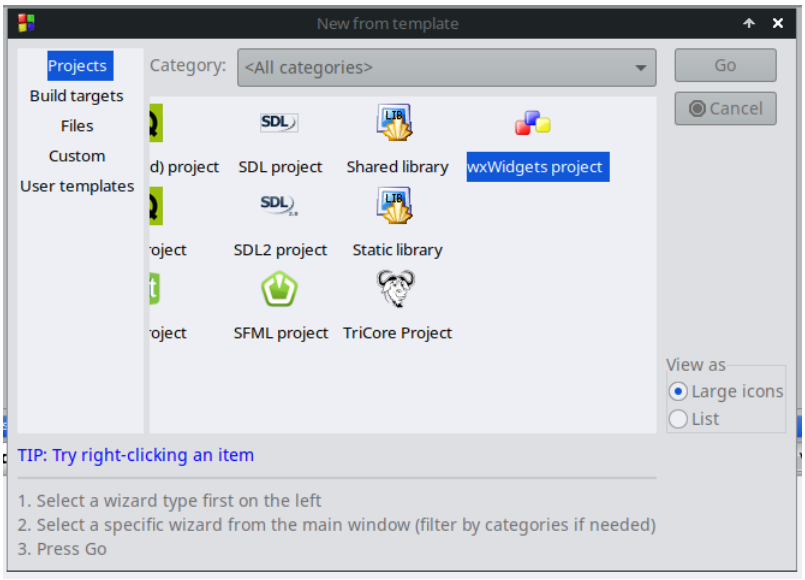
Se teniamo conto della situazione attualmente esistente per quanto riguarda il sistema operativo Mac e del fatto che ciò che s'ha da fare per lavorare sul sistema operativo Windows, oltre che essere laborioso, una volta funziona e tre no, dobbiamo concludere col dire che Code::Blocks, nella sua completezza, fornisce garanzia di funzionamento solo sul sistema operativo Linux, che è poi il sistema in cui è di casa C, C++ e tutto ciò che proviene da Unix.

4 Impostazione del progetto per programma con GUI

Mentre per produrre programmi semplici per console potremmo lavorare semplicemente su un file singolo senza creare un vero e proprio progetto, per sviluppare un programma dotato di interfaccia utente grafica (GUI) dobbiamo necessariamente impostare un progetto.

Lanciato Code::Blocks e con di fronte la finestra riprodotta a pagina 4 avviamo l'impostazione del nostro progetto da menu FILE ▷ NEW ▷ PROJECT.

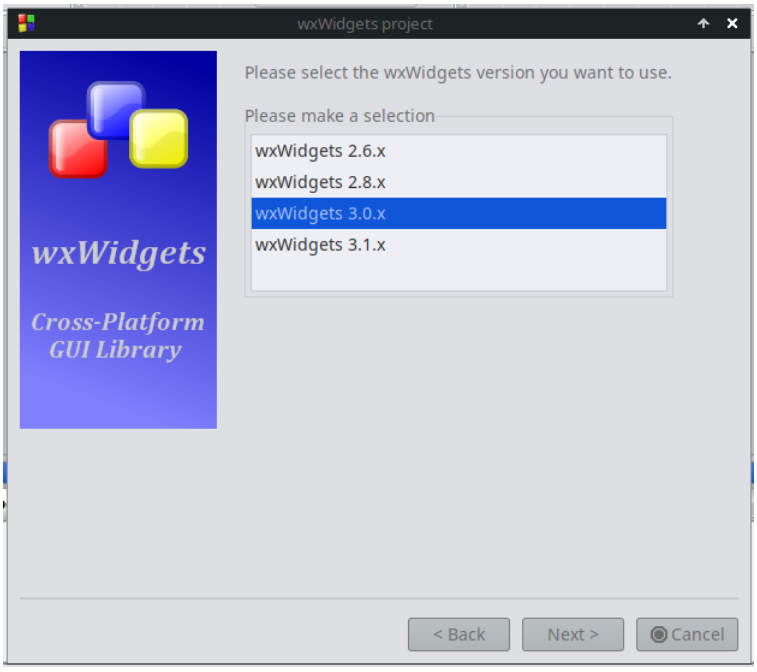
Nella finestra di dialogo che si apre



scorriamo fino a raggiungere la fine dell'elenco e clicchiamo sull'icona WXWIDGETS PROJECT, che l'ordine alfabetico relega all'ultimo posto, e poi sul pulsante GO in alto a destra.

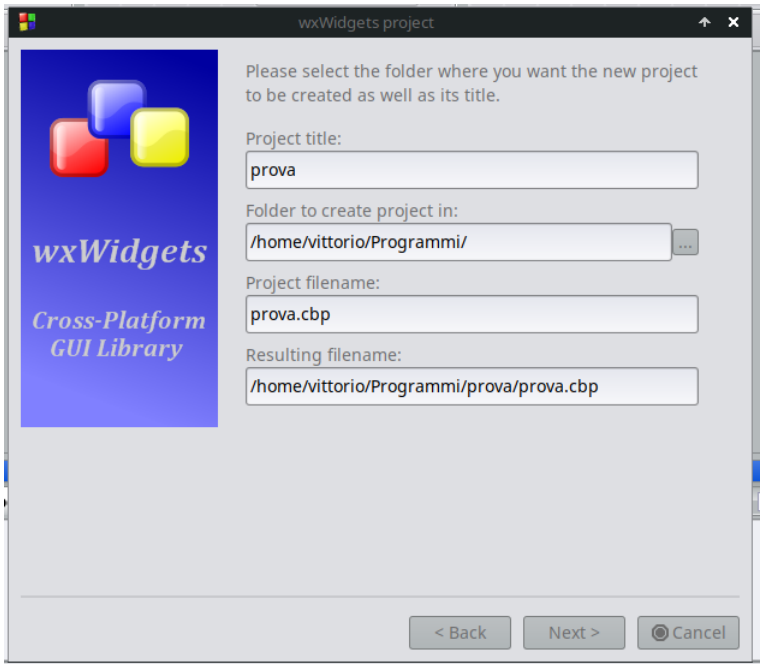
Si apre una inutile pagina di benvenuto, che potremmo fare in modo che non si ripresenti più e che superiamo cliccando su NEXT.

Nella successiva finestra di dialogo



selezioniamo la versione di wxWidgets da usare (nel mio caso è la 3.0) e clicchiamo su NEXT.

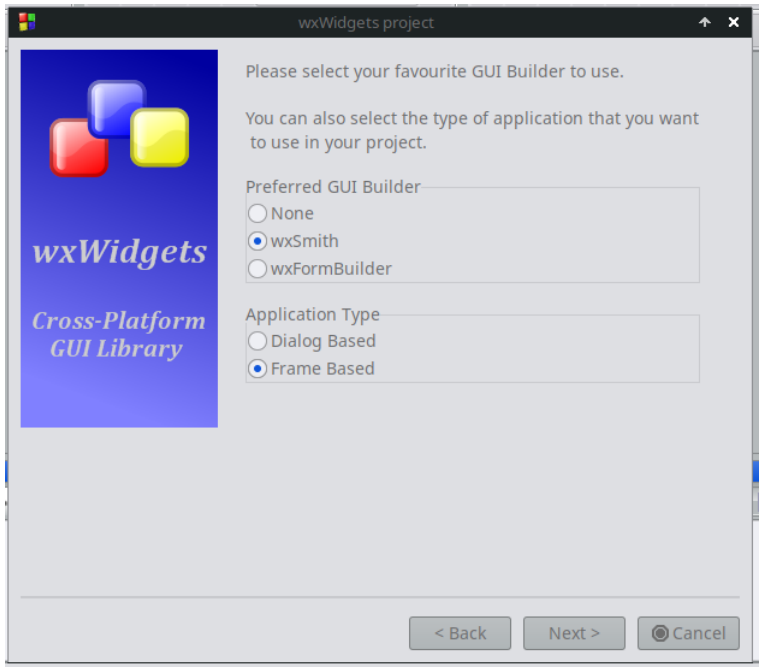
Nella finestra successiva dobbiamo dare un nome al progetto e collocarlo sul disco



Per la scelta del folder in cui ospitare il progetto sul disco siamo aiutati dall'apertura della finestra per la scelta del percorso che avviene cliccando sul pulsante a destra della finestrella.

Cliccando su NEXT ci troviamo di fronte ad una finestra di dialogo in cui ci si chiede di inserire il nostro nome di autore del progetto, la nostra email e il nostro indirizzo web. Ovviamente possiamo anche fare a meno di tutto ciò.

Cliccando su NEXT apriamo poi la seguente finestra

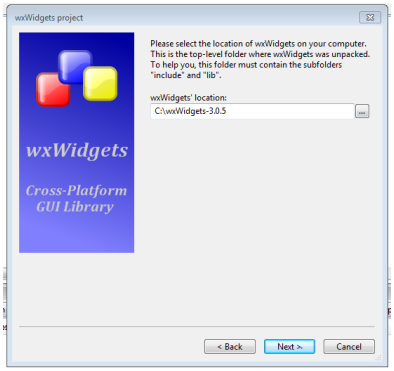


nella quale dobbiamo scegliere il designer per la nostra GUI (GUI builder), che, nel nostro caso è WXSMITH, e il tipo di applicazione che vogliamo costruire.

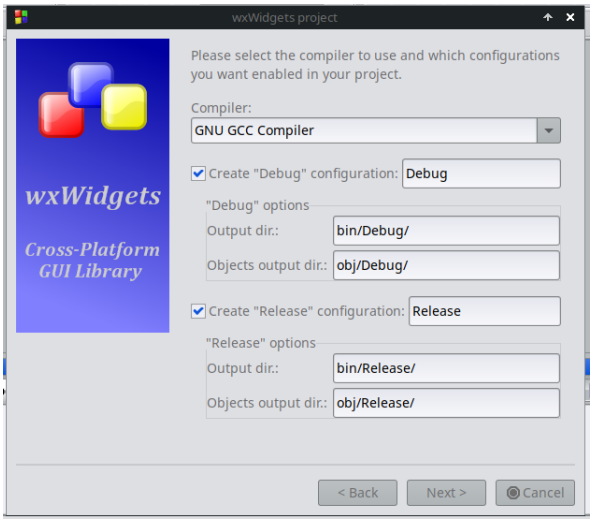
Possiamo scegliere tra una applicazione basata su finestre di dialogo volanti (DIALOG BASED) oppure basata sulla tradizionale finestra fissa sullo schermo del computer (FRAME BASED). Per questo tutorial sono stato sul tradizionale ed ho scelto quest'ultimo tipo di applicazione.

Per proseguire clicchiamo su NEXT.

A questo punto, solo se lavoriamo su Windows, ci viene chiesto di indicare dove si trovano le wxWidgets che dobbiamo utilizzare e lo facciamo aiutati dal click sul pulsante a destra della finestrella



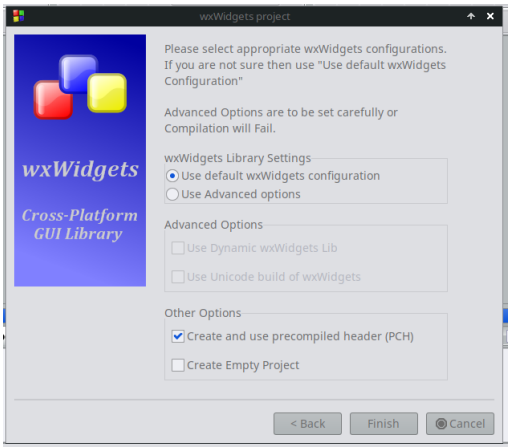
Successivamente abbiamo la finestra di settaggio del compilatore

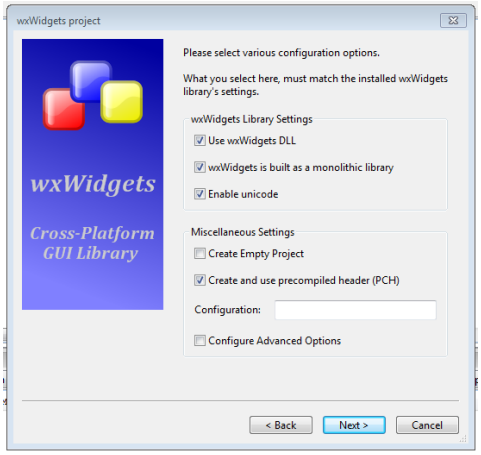


che si presenta già compilata da parte di Code::Blocks per annusamento. Da dilettanti potremmo deselezionare l'opzione CREATE "DEBUG" CONFIGURATION.

Se lavoriamo su Windows e abbiamo compilato le wxWidgets come abbiamo fatto prima, con l'opzione `BUILD=release`, dobbiamo proprio deselezionare l'opzione CREATE "DEBUG" CONFIGURATION.

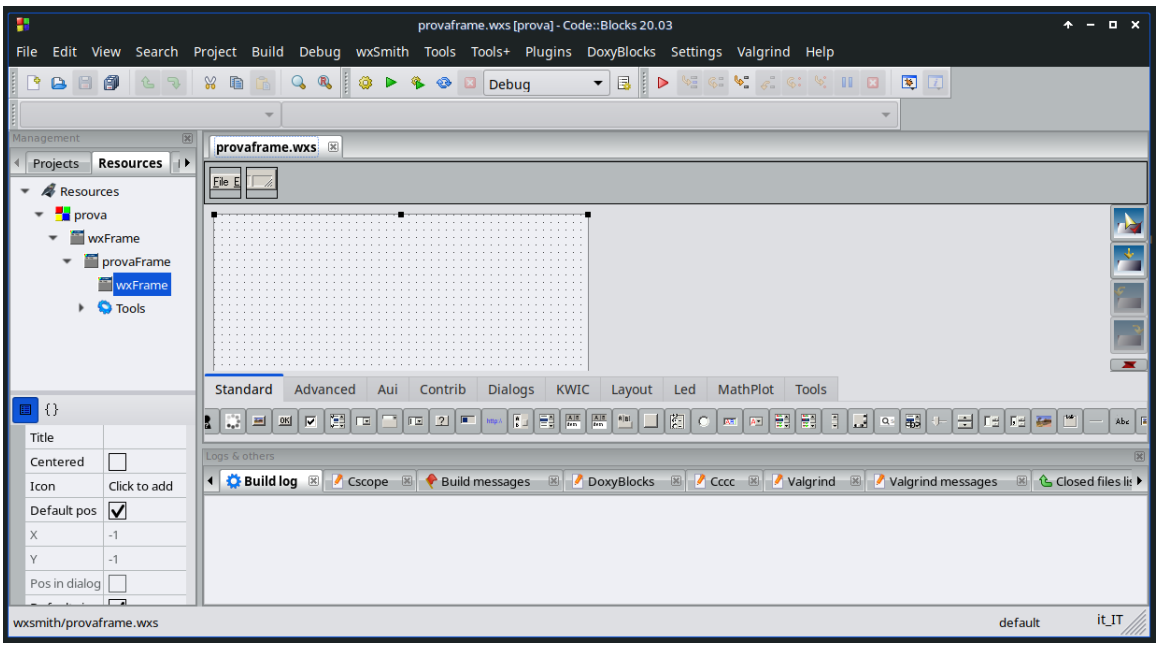
Cliccando su NEXT apriamo la finestra di configurazione delle wxWidgets





La finestra di sinistra è quella che si presenta se lavoriamo su Linux e la accettiamo praticamente come proposta.
La finestra di destra è quella che si presenta se lavoriamo su Windows e, avendo compilato le wxWidgets come abbiamo fatto prima, con l'opzione `MONOLITHIC=1` e l'opzione `UNICODE=1`, dobbiamo coerentemente selezionare, tra le altre, le relative opzioni di configurazione.

Arriviamo così, finalmente, alla finestra nella quale lavoreremo per costruire la nostra GUI.

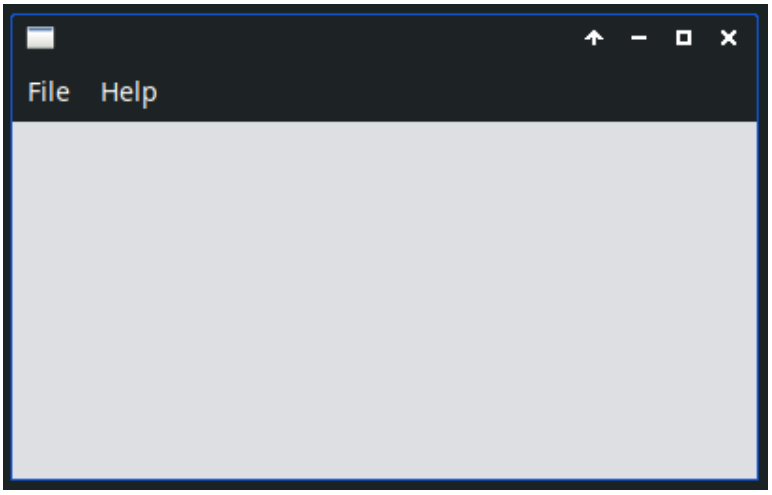


In realtà il primo passo è già compiuto in quanto, impostando il progetto, abbiamo già creato un programma che genera una finestra vuota.

Lo possiamo verificare compilando il progetto da menu BUILD ▷ BUILD o premendo il pulsante nella barra degli strumenti, eseguendolo poi da menu BUILD ▷ RUN o premendo il pulsante nella barra degli strumenti.

Cliccando preventivamente sulla tacca BUILD LOG nella zona inferiore della finestra siamo informati di come procede il tutto.

Se tutto ha funzionato avremo una finestra che, nell'impostazione del sistema a finestre del sistema operativo su cui sto lavorando (MX Linux, aspetto mx-comfort), appare così



Essa è già dotata di un menu. Aprendo FILE abbiamo accesso al comando QUIT per uscire dal programma e aprendo HELP abbiamo accesso ad una finestra informativa che ci dice con quale versione di wxWidgets è stato prodotto il programma.

L'eseguibile per produrla, che in Linux ha lo stesso nome del progetto senza estensioni e in Windows ha l'estensione `.exe`, lo troviamo nella directory dove era stato allocato il progetto (nel caso del tutorial che abbiamo appena visto `/home/vittorio/Programmi/prova`) nella sottodirectory `build/Debug` se abbiamo scelto di compilare con debug o nella directory `build/Release` se abbiamo scelto di evitare il debug.

Tutto il codice in linguaggio C++ per ottenere questa applicazione è stato scritto automaticamente da wxSmith e lo troviamo nella zona MANAGEMENT sulla sinistra della finestra di lavoro, nella figura aperta su RESOURCES, cliccando sulla linguetta PROJECTS.

5 Costruzione della GUI

La finestra di lavoro in cui produrremo la Gui ed il sottostante programma è quella riprodotta nella pagina precedente.

La parte superiore ha una barra dei menu autoesplicativa, sotto la quale si trova una barra degli strumenti nella quale sono rese accessibili attraverso click su icone le più frequenti occorrenze: passando il mouse sulle varie icone ne troviamo una descrizione.

Sotto, sulla destra, abbiamo una grande finestra che rappresenta il vero e proprio editor.

Nella parte inferiore della finestra abbiamo la barra dei wxWidgets che ci offre wxSmith.

Ancora sotto abbiamo una finestra nella quale possiamo aprire varie zone in cui Code::Blocks ci dà conto di ciò che sta facendo o ci manda segnali. La più interessante è la zona accessibile cliccando sulla linguetta BUILD LOG, nella quale ci viene descritto il procedimento di compilazione del programma, ci vengono segnalati eventuali errori che non rendono possibile la compilazione o ci viene confermato che la compilazione è avvenuta senza inconvenienti.

Sulla sinistra abbiamo la finestra MANAGEMENT, che è un browser di accesso ai file e alle risorse del progetto e del file system del computer.
Prescindendo da quest'ultima funzione, che normalmente non interessa per ciò che stiamo facendo, l'accesso ai file e alle risorse del progetto avviene in due zone diverse.
La zona corrispondente alla linguetta RESOURCES, quella aperta per default, mostra il nome degli oggetti grafici (widgets) utilizzati nel progetto.
La zona corrispondente alla linguetta PROJECTS mostra i file di codice relativi alla creazione ed al funzionamento di quegli oggetti e con doppio click sul nome di un file lo si apre nell'editor per poterlo modificare.
Se è aperta la zona RESOURCES, sotto al browser è presente una finestra di strumenti con cui possiamo intervenire per modificare le proprietà degli widgets (pulsante  attivato per default) o il tipo di evento collegato (pulsante ).
La selezione del widget per cui attivare questa finestra può avvenire selezionando il widget stesso nel browser delle risorse oppure cliccando (un solo click) sul widget che compare nella finestra disegnata nell'editor.
Con doppio click sul widget nella finestra dell'editor accediamo alla zona di codice relativa agli eventi collegati al widget nel file main del progetto.

* * *

Abbiamo visto che il processo di impostazione del progetto produce una finestra già dotata di un menu con alcune voci basiche e di una barra di stato.

Può darsi che non ci interessi avere un menu e una barra di stato nella GUI che andiamo a costruire.

Allora, prima di andare avanti per disegnare la GUI, eliminiamo queste cose.

A questo scopo, quando abbiamo esaurito il percorso che abbiamo visto nel Capitolo 4 per impostare il progetto, nella finestra MANAGEMENT/RESOURCES selezioniamo tra i tools la voce `wxMenuBar: MenuBar1` e la eliminiamo cliccando sul pulsante  che troviamo nella barra verticale sulla destra dell'editor. Lo stesso facciamo per la voce `wxStatusBar: StatusBar1`.
Per completezza, anche se non sarebbe strettamente necessario, cancelliamo anche questa parte del file ...Main.cpp

```
void provaFrame::OnQuit(wxCommandEvent& event)
{
    Close();
}

void provaFrame::OnAbout(wxCommandEvent& event)
{
    wxString msg = wxbuildinfo(long_f);
    wxMessageBox(msg, _("Welcome to..."));
}
```

A questo punto, eliminati menu e barra di stato, la finestra che produrrà il nostro programma sarà così

e da qui proseguiremo per costruire la nostra GUI senza menu.

5.1 Layout

Come ogni designer di GUI, anche wxSmith ci propone una serie di possibilità di gestire il collocamento dei widgets nella finestra, possibilità che possiamo riunire in due specie: collocamento relativo e collocamento assoluto.

Il collocamento relativo sistema i widgets in relazione a come sono collocati gli altri (sopra, sotto, di fianco, ecc.) secondo vari modelli e via via dimensiona la finestra della GUI automaticamente in relazione ai widgets che contiene.

Nel collocamento assoluto diamo noi una dimensione alla finestra e decidiamo noi dove collocare ciascun widget.

I modelli che ci offre wxSmith sono visibili cliccando sulla linguetta Layout nella barra dei widget che compare nella zona inferiore dell'editor quando è in modalità disegno, cioè quando è aperto sul file `.wxs` di wxSmith.

Personalmente non uso mai questo genere di automatismi che mi pare siano destinati a complicare cose semplici.

Anche in questo manualetto proporrò un metodo semplicissimo in cui non utilizzeremo alcuno di questi modelli.

Il lettore può comunque sperimentare in autonomia alla ricerca di qualche cosa che gli piaccia.

Basta provare questo e quello, anche aiutandosi con i pulsanti messi in verticale sulla destra dell'editor, tra i quali il penultimo dall'alto che ci consente di vedere i risultati di ciò che stiamo facendo.

Fino a quando non salviamo il file `.wxs` non abbiamo fatto nulla di definitivo e se chiudiamo il file `.wxs` senza salvarlo torniamo al progetto com'era prima delle nostre prove.

5.2 Widget indispensabili per una GUI

Se scorriamo la barra dei widget che vediamo nella parte bassa dell'editor aperto sul file `.wxs`, anche aprendo via via le varie zone contraddistinte dalle linguette, vediamo che gli oggetti grafici che ci offre wxWidgets sono moltissimi e ci possono consentire di programmare la GUI di progetti complicati come potrebbe essere quello di un foglio di calcolo tipo Excel o Calc.

Ma noi non abbiamo queste ambizioni e, comunque, magari anche per arrivare a fare cose di quel livello, dobbiamo prima sapere come funziona il tutto e lo possiamo vedere cominciando a fare cose semplici: è ciò che propongo in questo manualetto.

Per ora ci bastano i quattro oggetti che vedremo qui.

Prima ancora soffermiamoci però sull'oggetto principale che abbiamo costruito, più o meno dotato di menu, il **Frame**.

Una volta impostato il progetto lo vediamo disegnato nell'editor con aperto il file `.wxs` dove Code::Blocks ha automaticamente inserito, grazie al tool wxSmith, il codice che lo disegna.

Si tratta dell'oggetto principale perché è il contenitore della GUI che andremo a disegnare.

Su di esso, oltre a ciò che abbiamo già visto fare per eliminare il menu nel caso non ci interessi, possiamo intervenire per fare due cose molto importanti: stabilire un colore di sfondo per la nostra GUI e, visto che lavoreremo senza aiuti per il layout, dargli una dimensione grosso modo corrispondente a quella che avrà la GUI: grosso modo in quanto saremo sempre in tempo per rifiniture definitive.

Per scegliere il colore di sfondo, con selezionata lo voce `wxFrame` nella finestra RESOURCES del MANAGEMENT andiamo nella parte inferiore della finestra e scorriamo le Proprietà fino a trovare la voce BACKGROUND dalla quale accediamo alla possibilità di scegliere il colore di fondo del frame: cliccando a fianco della scritta BACKGROUND ci troviamo proposta una serie di sfondi in sfumature di grigio tra cui una casella CUSTOM da cui abbiamo accesso ad una finestra di dialogo per la scelta di un colore.

Il dimensionamento del frame lo otteniamo agendo sulle maniglie che vediamo nel contorno del frame che, almeno sugli schermi dei computer portatili, ci viene proposto in una dimensione che deborda dal basso della finestra dell'editor.

Ora vediamo i quattro oggetti con cui più ricorrentemente avremo a che fare per costruire una GUI.

5.2.1 wxStaticText

Corrisponde all'icona  nella barra dei widgets.

E' un'etichetta di testo che può servire per scrivere istruzioni per l'utente o per presentare i risultati di elaborazioni.

Si inserisce nel frame selezionando l'icona nella barra dei wigdets, trascinando sul frame riprodotto nell'editor e chiudendo con un click.

Si posiziona dove si vuole con le dimensioni più opportune agendo con il mouse e sulle maniglie.

Dalla finestra delle proprietà, proprietà LABEL, possiamo modificare la scritta (per default `Label`) o cancellarla lasciando in bianco l'etichetta.

Sempre nella finestra delle proprietà possiamo agire per il colore di sfondo (BACKGROUND), per il colore della scritta (FOREGROUND) e per il FONT. Per default il colore di sfondo viene ereditato da quello del frame contenitore. Abbiamo anche un sottomenu STYLE che ci può servire, tra l'altro, per stabilire l'allineamento della scritta.

Per modificare la scritta utilizzando il codice abbiamo a disposizione il metodo `SetLabel(<stringa>)`.

Dal momento che l'argomento di questa funzione deve essere una stringa, se vogliamo che nella nostra etichetta venga scritto un dato di tipo numerico richiamando il nome della variabile che lo contiene dobbiamo procedere al casting ed un modo semplice per farlo può essere questo:

. sia `x` la variabile di tipo numerico,

. definiamo una variabile di tipo stringa con

```
wxString xs
```

. trasferiamo il contenuto della variabile `x` in `xs` con

```
xs << x
```

ed ora possiamo passare a SetLabel l'argomento `xs` per scrivere nell'etichetta il valore di tipo numerico della variabile `x`.

5.2.2 wxTextCtrl

Corrisponde all'icona  nella barra dei widgets.

E' il luogo della GUI per l'input dei dati che il programma dovrà elaborare e si presenta come una finestrella di una sola riga.

Lo sistemiamo nel frame con il solito procedimento.

Per default contiene la scritta `Text` allineata a sinistra.

Agendo nella finestra delle proprietà è bene eliminare la scritta, agendo sulla proprietà TEXT, in modo che la finestra risulti vuota e per l'allineamento possiamo modificarlo agendo sotto la voce STYLE.

L'estrazione del valore immesso nella finestrella avviene con il metodo
`GetValue()`
e il valore estratto è di tipo stringa.

Se dobbiamo utilizzare questo valore per fare calcoli occorre procedere al casting passando il valore estratto al metodo `wxAtoi()`, per avere un valore di tipo `int`, o al metodo `wxAtof()`, per avere un valore di tipo `float`.

5.2.3 wxButton

Corrisponde all'icona  nella barra dei widgets.

E' il classico pulsante che nelle GUI costituisce il principale, spesso l'unico, strumento per ottenere qualche accadimento.

Lo sistemiamo nel frame con il solito procedimento e nella finestra delle proprietà troviamo gli strumenti per modificare la scritta sopra di esso (voce LABEL), per modificarne il colore di sfondo (voce BACKGROUND), per modificare il colore della scritta sopra di esso (voce FOREGROUND).

L'evento che per default è associato al pulsante è `OnButtonClick`, cioè il click su di esso con il pulsante sinistro del mouse.

Con doppio click sul pulsante inserito nel frame apriamo il file `Main.cpp` del progetto con il cursore lampeggiante nella zona dove inserire il codice da collegare all'evento.

5.2.4 wxPanel

Corrisponde all'icona  nella barra dei widgets.

E' un pannello che può contenere altri widget e che può servire per dare colorazione diversa a certe zone del frame.

Il colore del suo sfondo lo possiamo stabilire con lo stesso procedimento che abbiamo visto per il colore di sfondo del frame, applicato selezionando la voce `wxPanel` nella finestra RESOURCES del MANAGEMENT.

Possiamo anche dare un colore al pannello utilizzando il codice con il metodo
`SetBackgroundColour(wxColour(<combinazione_RGB>))`.

* * *

Se lavoriamo senza utilizzare strumenti per gestire il layout, il primo widget inserito nel frame lo occupa completamente e tutto si sistema con l'inserimento del secondo widget. Se però il primo widget inserito è wxPanel, questo inserimento copre tutto il frame in modo definitivo, sostituendosi ad esso come contenitore e rendendone le dimensioni non più modificabili. E' pertanto da evitare di inserire il pannello come primo widget.

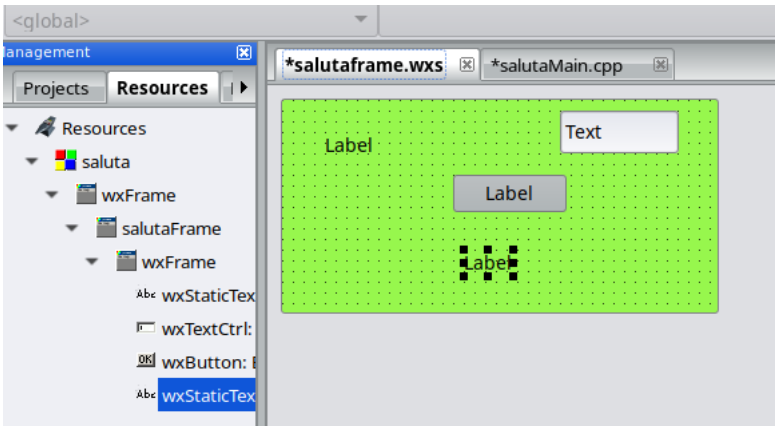
6 Alcuni esempi

Cominciamo da un programmino che chiede il nome all'utente per salutarlo.

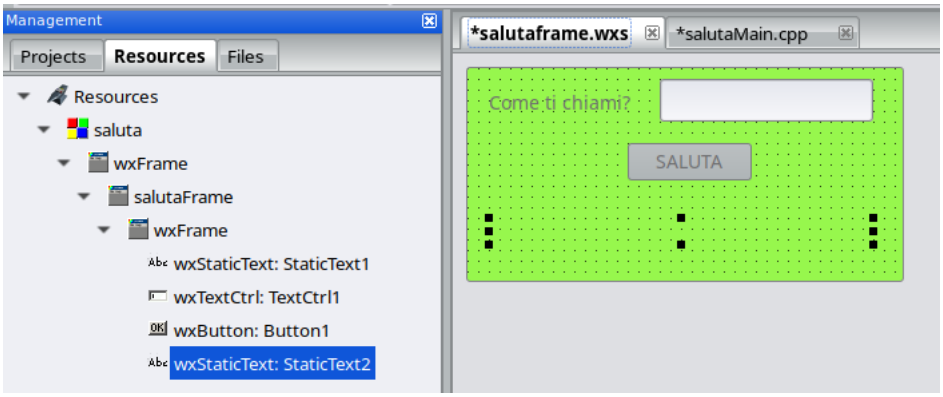
Impostiamo il programma, come abbiamo visto fare nel Capitolo 4, chiamandolo `saluto` ed eliminiamo il menu, che qui non ci serve, come abbiamo visto fare alla fine del Capitolo 5.

Ridimensionamo il frame, gli diamo un titolo (SALUTO) agendo nella finestrella TITLE delle Proprietà, gli diamo un colore che lo renda un tantino gradevole agendo nella finestrella

BACKGROUND delle Proprietà e vi collochiamo i widget che ci serviranno: due wxStaticText (uno per chiedere all'utente cosa fare e uno per formulare il saluto), un wxTextCtrl (per dar modo all'utente di inserire il proprio nome) e un wxButton (cliccando sul quale otteniamo di essere salutati)



Rifiniamo il tutto, collocando a dovere i widget e sistemando le scritte che li riguardano, le loro dimensioni, ecc., come visto fare nel Paragrafo 5.2, in modo da ottenere una GUI come questa



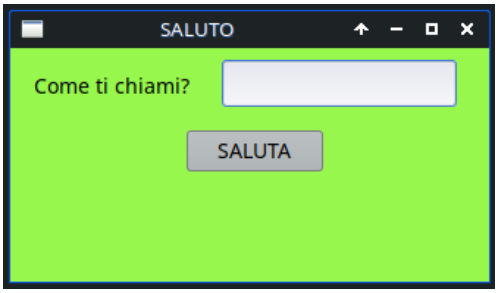
Nella finestra RESOURCES abbiamo l'elenco dei widgets inseriti nel frame con indicato il nome loro attribuito come oggetti: `StaticText1` per il wxStaticText collocato in alto a sinistra, `TextCtrl1` per il wxTextCtrl, `Button1` per il wxButton e `StaticText2` per il wxStaticText in basso, quello selezionato, in modo che se ne veda la dimensione allungata destinata a contenere il messaggio di saluto. La `TextCtrl1` è stata predisposta per allineare a destra il nome inserito. La `StaticText2` è stata predisposta per ospitare una scritta abbastanza lunga, per allineare al centro la scritta e per rendere questa scritta in colore rosso usando il font `Caladea Bold Italic` corpo 20.

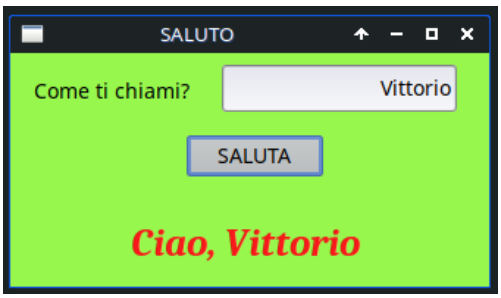
Ora con doppio click sul pulsante Saluta apriamo nell'editor il file `salutaMain.cpp` e vediamo il cursore lampeggiare in corrispondenza del codice per gestire l'evento `OnButton1Click`.

Inseriamo il codice che vediamo in colore rosso tra le parentesi graffe che troviamo già predisposte per ospitarlo.

```
void salutaFrame::OnButton1Click(wxCommandEvent& event)
{
wxString nome;
nome = TextCtrl1->GetValue();
StaticText2->SetLabel("Ciao, " + nome);
}
```

Salviamo i file, compiliamo il programma e lo utilizziamo con questi risultati





La finestra di sinistra rappresenta la GUI al lancio del programma. la finestra di destra rappresenta la GUI dopo che è stato inserito il nome e si è cliccato sul pulsante SALUTA.

* * *

Ora un programma che ci mostra il colore corrispondente ad una certa combinazione dei colori Rosso, Verde e Blu, il modello additivo (chiamato RGB, dalle iniziali dei nomi inglesi dei tre colori) che è stato quasi universalmente adottato per la rappresentazione e la visualizzazione di immagini in dispositivi elettronici.

La quantità di ciascuno di questi colori nel mix si indica con un numero intero compreso tra 0 e 255.

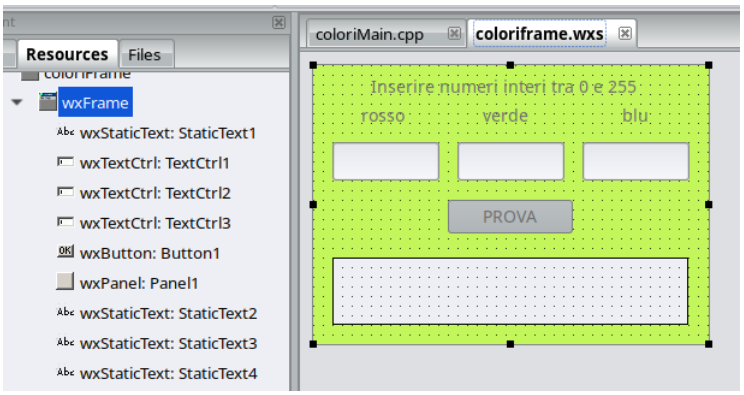
Il mix con la quantità massima di tutti i tre colori (255, 255, 255) corrisponde al bianco.

Il mix con nessuno dei tre colori (0, 0, 0) corrisponde al nero.

Per vedere cosa corrisponde a tutte le altre combinazioni possibili costruiamo questo programmino.

Ci servono quattro wxStaticText per le indicazioni da fornire all'utente, tre wxTextCtrl per inserire il numero intero indicante la quantità di ciascun colore, un wxButton per dare il via alla combinazione e un wxPanel per visualizzare il colore risultante dalla combinazione.

Con le tecniche ormai ampiamente descritte impostiamo un progetto intitolato Colori, eliminiamo il menu che non ci serve, ridimensioniamo il frame, gli diamo un titolo (PROVA COLORE), un gradevole colore di fondo, aggiungiamo e sistemiamo i necessari widget in modo da ottenere una GUI di questo tipo

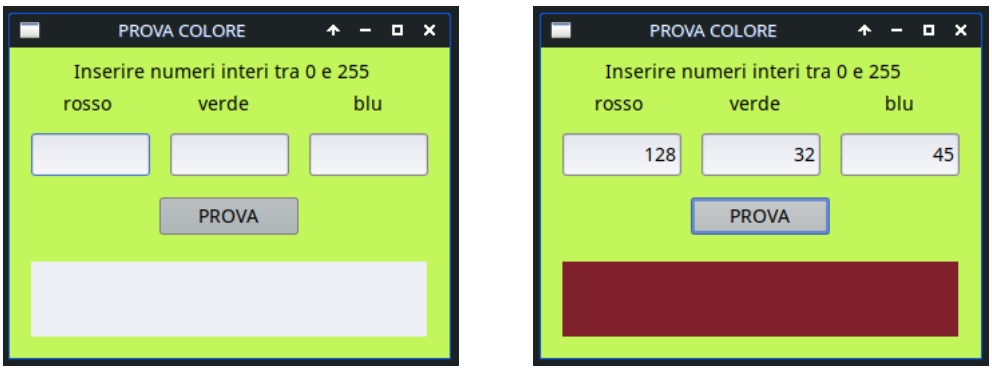


Ora con doppio click sul pulsante Prova apriamo nell'editor il file `coloriMain.cpp` e vediamo il cursore lampeggiare in corrispondenza del codice per gestire l'evento `OnButton1Click`.

Inseriamo il codice che vediamo in colore rosso tra le parentesi graffe che troviamo già predisposte per ospitarlo.

```
void coloriFrame::OnButton1Click(wxCommandEvent& event)
{
    int r, g, b;
    r = wxAtoi(TextCtrl1->GetValue());
    g = wxAtoi(TextCtrl2->GetValue());
    b = wxAtoi(TextCtrl3->GetValue());
    Panel1->SetBackgroundColour(wxColor(r,g,b));
}
```

Salviamo i file, compiliamo il programma e lo utilizziamo con questi risultati

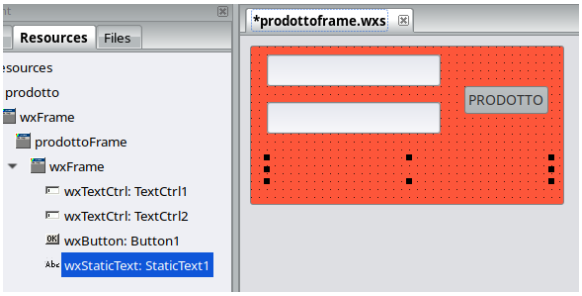


La finestra di sinistra rappresenta la GUI al lancio del programma. la finestra di destra rappresenta la GUI dopo che è stata inserita la combinazione di colori 128, 32, 45 e si è cliccato sul pulsante PROVA. Il colore che vediamo sotto è quello corrispondente alla combinazione inserita.

* * *

Ora un esempio che nulla aggiunge alla tecnica per costruire la GUI ma che serve per illustrare alcuni trucchi quando abbiamo a che fare con programmi di calcolo vestiti da wxWidgets.

Con il solito procedimento impostiamo un progetto che costruisce la GUI per un programma che calcola il prodotto tra due numeri



Ora dobbiamo scrivere il codice da abbinare all'evento click sul pulsante PRODOTTO affinché nella `StaticText1` sia visualizzato il prodotto tra i due numeri inseriti in `TextCtrl1` e `TextCtrl2`.

La prima complicazione deriva dal fatto che i valori che leggiamo nelle due caselle di input sono di tipo stringa, il prodotto dobbiamo calcolarlo su valori numerici e il risultato che dobbiamo inserire nella casella di output deve essere di tipo stringa.

Per il casting da stringa a intero o a float dei valori in input abbiamo a disposizione le funzioni `wxAtoi()` e `wxAtof()`.

Il casting da valore numerico a stringa wxWidgets ci consente di farlo semplicemente utilizzando l'operatore << tra la variabile stringa in cui immettere il valore numerico e la variabile di tipo numerico che lo contiene.

Sicché, per esempio, il codice per calcolare il prodotto tra due interi collegato all'evento click sul pulsante `wxButton1` potrebbe essere il seguente, scritto in colore rosso:

```
void prodottoFrame::OnButton1Click(wxCommandEvent& event)
{
int x, y, p;
wxString prodotto;
x = wxAtoi(TextCtrl1 -> GetValue());
y = wxAtoi(TextCtrl2 -> GetValue());
p = x * y;
prodotto << p;
StaticText1 -> SetLabel(prodotto);
}
```

Purtroppo il tipo `int` non ci consente di ottenere risultati validi se il valore del prodotto supera di poco i due miliardi (2.147.483.647).

Visto che la `wxAtoi()` funziona anche per il tipo `long`, se le variabili `x`, `y` e `p` le dichiariamo di tipo `long` anziché `int`, possiamo ottenere risultati validi fino a valori espressi con 18 cifre.

Per operare su tipi a virgola mobile possiamo dichiarare le variabili `x`, `y` e `p` per il calcolo di tipo `float` e convertirle da stringa con la funzione `wxAtof()`.

Ma con la precisione a 6 cifre del tipo `float` non andremmo lontano con i nostri calcoli e dobbiamo complicare un tantino il nostro codice. Né potremmo rimediare dichiarando di tipo double le variabili di input.

Fortunatamente wxWidgets ci offre una soluzione con la funzione di casting, membro degli oggetti di tipo stringa, `toCDouble()` e con la funzione `wxNumberFormatter::ToString()`.

Quest'ultima accetta due parametri: il primo è il nome della variabile che contiene il valore numerico da formattare come stringa e il secondo è il numero di cifre decimali da esporre.

Per utilizzare questa funzione dobbiamo includere l'header che la contiene nel nostro programma e lo facciamo inserendo, all'inizio del file Main, nella zona dove ci sono gli `#include` la voce

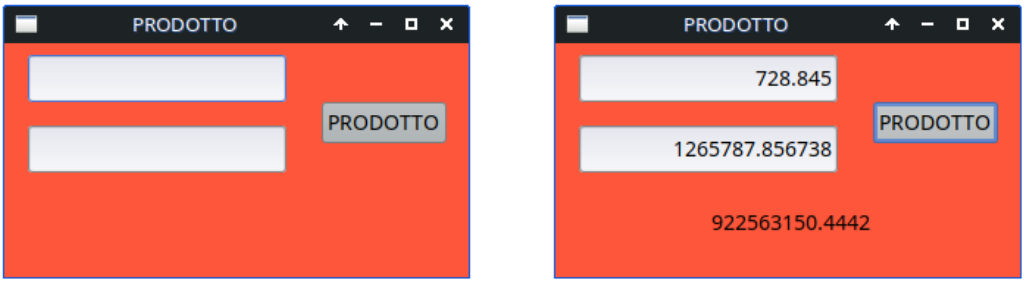
```
#include <wx/numformatter.h>.
```

Il codice per calcolare il prodotto tra due numeri collegato all'evento click sul pulsante `wxButton1` potrebbe ora essere il seguente, scritto in colore rosso:

```
void prodottoFrame::OnButton1Click(wxCommandEvent& event)
{
wxString x, y, p;
x = TextCtrl1->GetValue();
y = TextCtrl2->GetValue();
double xx, yy, pp;
x.ToCDouble(&xx);
y.ToCDouble(&yy);
pp = xx * yy;
int decimali = 4;
p = wxNumberFormatter::ToString(pp, decimali);
StaticText1->SetLabel(p);
}
```

Con questo programma possiamo calcolare il prodotto di due numeri decimali, di un intero e di un decimale o anche di due interi e il risultato verrà esposto con quattro cifre decimali in precisione double (15 cifre precise).

Nella seguente figura, l'immagine di sinistra rappresenta la GUI al lancio del programma e l'immagine di destra rappresenta la GUI dopo che sono stati inseriti i due numeri e si è premuto il pulsante PRODOTTO



* * *

Per finire, una piccola applicazione che può essere utile per soddisfare una strana esigenza.

L'esigenza è quella di avere un file di testo che possa essere letto e modificato soltanto da me ma possa essere alimentato, senza poterlo leggere, da chiunque abbia accesso al mio computer senza conoscere la mia password, semplicemente perché il computer è acceso.

Una cosa del genere può essere fatta agevolmente su Linux.

Innanzi tutto apro il file manager come super-utente e creo il file, che chiamo `appunti`, in un punto di rapido accesso, per esempio sul desktop.

Sempre da file manager come super-utente ne stabilisco le proprietà in modo che siano queste

Ora creo l'applicazione per alimentarlo.

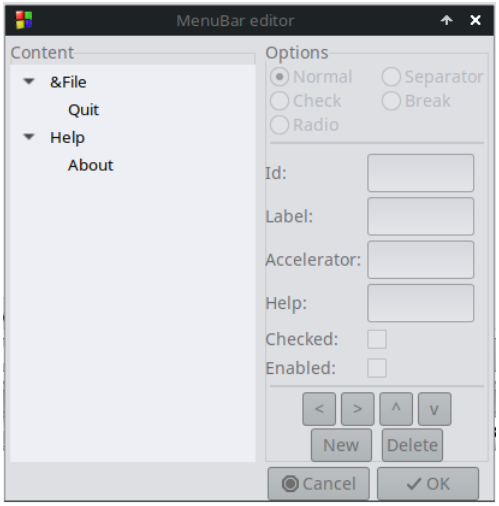
Imposto un progetto come abbiamo visto fare nel Capitolo 4 e lo chiamo `appunti`.

In questo progetto mantengo il menu e la barra di stato.

Al menu aggiungo una voce che dia modo di salvare gli appunti che verranno scritti nella nostra applicazione.

Nella finestra di lavoro, nella parte superiore dell'editor aperto sul file `appuntiframe.wxs`, compare l'icona del tool Menu  .

Con doppio click su questa icona apro l'editor del menu



Mi posiziono all'interno della voce `&File` selezionando la voce `Quit` che vi è già inserita e clicco sul pulsante NEW.

Nella finestra successiva, inserisco la dicitura `Salva` nella finestrella LABEL e la dicitura `Salva appunto` nella finestrella Help.

Con selezionata la nuova voce `Salva` agendo con il pulsante freccia su ^ la sposto sopra la voce `Quit`.

Cliccando su OK salvo il nuovo menu comprensivo della nuova voce SALVA.

Ora, nella finestra MANAGEMENT/RESOURCES apro la voce TOOLS e via via quelle ivi contenute fino a poter selezionare la voce `Salva` in modo da aprirne la finestra delle proprietà.

Con la voce Salva selezionata apro la casella degli eventi, contrassegnata dall'icona {}, clicco sulla finestra di fianco alla dicitura EVT_MENU, dove c'è scritto `-None-`, apro il menu, clicco su ADD NEW HANDLER e accetto quanto viene proposto nella finestra di dialogo che si apre.

Ciò crea il seguente codice nel file `appuntiMain.cpp`:

```
void appuntiFrame::OnMenuItem3Selected(wxCommandEvent& event)
{
}
```

e tra le parentesi graffe inserirò il codice da eseguire quando verrà selezionata la voce di menu SALVA.

A questo punto occorre inserire nella GUI un widget per acquisire il testo degli appunti.

Innanzi tutto riduco il frame a una dimensione ragionevole per ospitare brevi appunti e lo chiamo APPUNTI agendo in corrispondenza della voce TITLE.

Per acquisire il testo abbiamo il widget wxTextCtrl, che, nella sua impostazione di default serve per l'inserimento di una sola riga di testo (e così lo abbiamo utilizzato nei precedenti esempi). Con una piccola modifica lo adatto in modo che ci si possano inserire più righe e che si vada a capo dopo che è stata raggiunta la dimensione orizzontale della finestra di editing.

Prima di tutto seleziono questo widget e lo inserisco nel frame.

Poi, nella finestra delle proprietà, elimino la dicitura `Text` che compare nella proprietà TEXT e, aperto il menu della proprietà STYLE, seleziono le voci WXTE_MULTILINE e WXTE_WORDWRAP.

Ora salvo tutto cliccando sull'icona  nella barra degli strumenti.

A questo punto non resta che inserire il codice che faccia funzionare l'applicazione, cioè che consenta di scrivere un appunto e di inserirlo nel file appunti senza che chi agisce ne possa vedere il contenuto (ciò che succederebbe se costui conoscesse la mia password e utilizzasse un semplice editor di testo).

Per fare questo dobbiamo ricorrere alla libreria C++ fstream, che importo nel progetto scrivendo nel file appuntiMain.cpp, all'inizio nella zona degli `#include`, in coda a quelli già esistenti, l'istruzione

```
#include <fstream>;
```

Infine inseriamo il codice qui scritto in rosso tra le parentesi graffe nella zona predisposta per contenere il codice di ciò che si deve fare alla selezione della voce di menu SALVA:

```
void appuntiFrame::OnMenuItem3Selected(wxCommandEvent& event)
{
using namespace std;
wxString testo = TextCtrl1->GetValue();
fstream f("/home/vittorio/Scrivania/appunti", ios::out|ios::app);
f << testo;
f.close();
}
```

Ultimissima pennellata può essere quella di fare in modo che la nostra applicazione parli italiano. Infatti, la parte di menu che ho fatto io è in italiano ma quella che ha predisposto Code::Blocks è in inglese.

La parte predisposta da Code::Blocks è stabilmente modificabile e le variazioni apportate ad essa possono essere salvate solo se poi non interveniamo più sul progetto.

Ecco perché faccio questa operazione per ultima.

Con il file `provaMain.cpp` aperto nell'editor individuo la zona

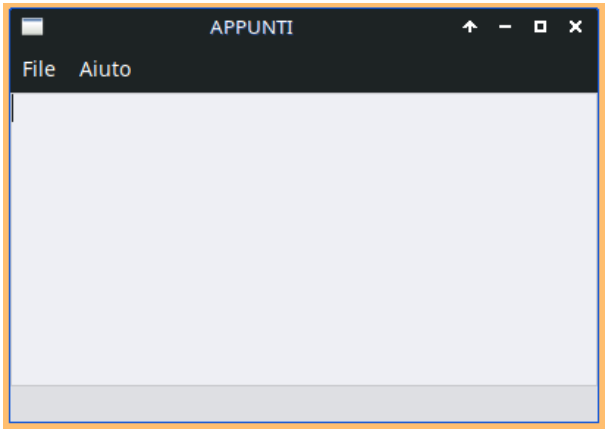
```
Menu1 = new wxMenu();
MenuItem1 = new wxMenuItem(Menu1, idMenuQuit, _("Esci\tAlt-F4"), _("Lascia l'applicazione"), wxITEM_NORMAL);
Menu1->Append(MenuItem1);
MenuBar1->Append(Menu1, _("&File"));
Menu2 = new wxMenu();
MenuItem2 = new wxMenuItem(Menu2, idMenuAbout, _("About\tF1"), _("Mostra informazioni sull'applicazione"), wxITEM_NORMAL);
Menu2->Append(MenuItem2);
MenuBar1->Append(Menu2, _("Aiuto"));
```

e sostituisco le scritte in rosso, in italiano, a quelle di prima, che erano in inglese.

Così modifico la zona

```
void appuntiFrame::OnAbout(wxCommandEvent& event)
{
wxString msg = wxbuildinfo(long_f);
wxMessageBox(msg, _("Benvenuto..."));
}
```

Compilato il progetto, posso disporre di questa applicazione



con la quale chiunque abbia accesso al mio computer può scrivere un appunto da inserire nel file protetto `appunti` che è possibile vedere solo conoscendo la mia password.

Basta scrivere l'appunto nella finestra bianca, scegliere FILE ▷ SALVA e chiudere con FILE ▷ ESCI.

Per vedere e modificare il file con il mio editor di testo preferito mi basta scrivere questo comando a terminale

`sudo` <nome_editor_di_testo> `/home/vittorio/Scrivania/appunti`

e inserire la mia password.

7 Conclusioni

Penso che questo manualetto sia servito per evidenziare i grandi pregi di Code::Blocks e, nel caso dei programmi con GUI, del suo plugin wxSmith.

Anche con i semplici esempi che ho proposto è tuttavia del tutto evidente che Code::Blocks è un valido aiuto ma non può certo servire a chi non conosca i non facili linguaggi C e C++ e, per chi utilizza le wxWidgets, le complicate sintassi che vi si ricollegano.

Soprattutto se vogliamo fare qualche cosa di più complicato rispetto agli esempi che ho proposto.