

Pemrograman Multiplatform dengan WxWindows

Bagian 1 dari 2 Artikel

C++ adalah salah satu bahasa berorientasi objek yang matang dan telah digunakan dalam pembangunan banyak aplikasi skala besar. KDE adalah salah satu contoh yang baik. Saat ini, untuk membangun aplikasi GUI, kita bisa mempergunakan beberapa pustaka GUI yang terkenal, salah satunya adalah WxWindows atau untuk lebih singkatnya, wx.

Percaya atau tidak, segala sesuatu yang lebih mudah atau lebih modern selalu memiliki efek samping yang terkadang cukup mahal. Mari kita tarik lebih spesifik ke dalam pengembangan aplikasi. Penulis telah menggunakan Python selama kurang lebih tiga tahun dan menemukan Python adalah bahasa pemrograman top yang luar biasa. Membuat segala sesuatu dapat dilakukan dengan mudah menggunakan Python. Apalagi dengan berbagai tipe data canggih di Python, membangun aplikasi akan terasa jauh lebih mudah.

Sebagai contoh, dengan panjang string yang secara teori tidak terbatas di Python, dan dilengkapi dengan pemrosesan string yang super mudah, membangun aplikasi yang berhubungan dengan *text processing* sama sekali bukan masalah.

Sayangnya, untuk aplikasi yang lebih serius, penulis tidak berani menggunakan Python. Yang pertama adalah bahwa kemampuan python dalam menangani aplikasi skala besar. Tidaklah bijak untuk menempatkan Python untuk membangun core sebuah aplikasi besar. Bagaimanapun, sistem kerja Python yang membaca baris demi baris dan semuanya dikerjakan oleh satu interpreter (seberapa canggih pun) dalam banyak kasus tidak cocok digunakan untuk membangun *core* aplikasi kompleks.

Yang kedua adalah ukuran kecepatan. Python sangatlah lambat dan sangat pantas dipertandingkan dengan Java dalam soal kelambatan. Yang ini memang dibayar dengan kemudahan dan kecepatan pemem-

bangun aplikasi.

Yang ketiga adalah ukuran interpreter Python. Dalam sistem yang sensitif terhadap space, tidaklah bijak untuk menggunakan python. Interpreter Python, modul, dan segala pustaka yang diperlukan bisa berukuran belasan sampai puluhan MB. Walaupun dengan melakukan kompilasi atau *freezing* sekalipun, ukuran *executable* tetap besar dan lambat.

Sejak beberapa waktu yang lalu, penulis berusaha untuk melakukan pendekatan yang menurut penulis cukup arif, setidaknya bagi penulis sendiri. Ketika akan membangun aplikasi yang potensial menjadi besar dan kompleks, penulis menggunakan C++ sebagai bahasa pemrograman untuk menulis core aplikasi.

Setelah core selesai, di masa depan, fleksibilitas akan ditambahkan kepada aplikasi tersebut sehingga dapat bekerja secara modular. Dan, modul-modul dalam aplikasi tersebut akan ditulis dengan bahasa Python. Dengan demikian, setidaknya, penulis mencoba untuk melakukan pendekatan kompromistis yang memiliki risiko paling rendah. Kalaupun menghadapi masalah *space* atau *time*, beberapa modul kritis masih bisa ditulis dengan C++. Namun, sebisa mungkin, tidak semua modul ditulis dengan C++ karena akan menjadi sangat mahal.

Harus disadari bahwa C++ adalah bahasa yang cukup tua. Dan, dalam beberapa hal, cukup ketinggalan zaman. Namun, kematangannya tidak diragukan lagi. Wa-

laupun, semua keunggulan tersebut harus dibayarkan dengan segala kerumitan yang harus dihadapi oleh sang Developer.

Di Linux, C++ mendapat perhatian yang sangat besar. Puluhan pustaka kelas tinggi siap digunakan. Pustaka GUI pun tidak kalah banyak. Salah satu pustaka GUI (sangat lengkap) yang populer adalah Qt. Kepopuleran Qt terus bertambah dengan digunakannya Qt pada pembuatan KDE.

Sayangnya, Qt memiliki satu kelemahan yang menurut penulis cukup mengganggu. Lisensinya. Untuk menulis program *free software*, Qt versi *free* memang bisa digunakan. Namun, ketika kita ingin menulis aplikasi *proprietary* (Qt versi *free* hanya bisa digunakan untuk menulis aplikasi *free* juga), kita harus membeli Qt versi bayarnya, yang versi enterprenisnya berharga ribuan dollar. Walaupun hal ini bisa diselesaikan dengan membuat *free software* atau membeli Qt versi bayar, bagi penulis, ada satu keterbatasan yang menghadang, dan di masa depan, keterbatasan ini bisa menjadi lebih besar.

Selain Qt, kita mengenal pula pustaka WxWindows, yang belum lama ini berubah menjadi wxWidgets karena Microsoft keberatan dengan penggunaan istilah Windows di dalam nama pustaka ini. Namun, karena istilah wxWindows telah digunakan cukup lama, banyak pengguna yang masih menggunakan istilah ini. Umumnya, agar lebih singkat, istilah wxWindows seringkali disebut menjadi wx saja. Lisensi wx sangat fleksibel sehingga kita dapat menulis aplikasi *free* ataupun *proprietary* menggunakan

wx. Dan, itu sepenuhnya legal.

Menggunakan wx, kita bisa membangun aplikasi GUI kelas tinggi. Wx sendiri pada kenyataannya tidak hanya mengkhususkan diri pada masalah GUI. Threading, Networking, dan struktur-struktur data lain seperti penanganan string juga disertakan. Dengan demikian, menggunakan wx, kita bisa membangun aplikasi GUI menggunakan bahasa C++ dengan cara yang mudah, dan aplikasi kita nantinya dapat dikompilasi ulang di berbagai platform dengan berbagai kompilator yang tersedia.

Salah satu keunggulan wx adalah dapat menerima berbagai macam kompilator, termasuk Borland C++, Microsoft Visual C++, Watcom C++, Metrowerks CodeWarrior, GNU C++, dan lain sebagainya. Bahkan, wx rela mengorbankan untuk tidak menggunakan template C++ karena banyak kompilator yang tidak mendukung sistem template. Walau, di masa depan, penggunaan template akan terus didorong.

Saat ini, pengembangan aplikasi berbasis GUI harus diperhatikan benar-benar karena perkembangan GUI tersebut berlangsung cepat. Dan, terkadang, banyak pula client yang ingin membangun semua aplikasinya berbasis web. Mungkin karena benar-benar butuh, mungkin karena ikut-ikutan.

Dalam pengembangan aplikasi GUI, salah menulis kode saja, atau salah memilih pustaka saja, bisa-bisa kode-kode kita menjadi ketinggalan zaman dan tidak dapat dipergunakan kembali.

Sebelum kita melangkah lebih jauh ke wx, kita perlu melihat apakah wx cocok untuk digunakan sebagai pustaka untuk membangun aplikasi yang Anda butuhkan. Pertama-tama, wx tidak selalu cocok untuk membangun setiap aplikasi. Apabila Anda membangun aplikasi yang sangat OLE intensive di Windows, maka penggunaan wx tidaklah tepat. Lebih baik menggunakan pendekatan Visual Basic. Yang kedua, apabila Anda ingin menulis daemon kelas tinggi dengan memanfaatkan sistem thread dan pustaka network yang dimiliki oleh wx. Untuk kebutuhan tersebut, Anda tidak perlu menggunakan wx.

Namun, apabila Anda ingin membangun aplikasi GUI yang berjalan pada berbagai platform, maka wx cocok digunakan. Menggunakan wx, Anda bisa mempergu-

nakan berbagai widget modern yang tidak kalah dengan implementasi bahasa lain. Sebagai contoh, wx dilengkapi dengan pustaka wxGrid, yang bisa Anda pergunakan untuk membangun spreadsheet sendiri.

Selain itu, wx juga dapat menghasilkan aplikasi dengan look and feel native untuk setiap platform. Hal ini berarti, ketika aplikasi Anda dikompilasi dan dijalankan di Windows, maka tampilannya akan mirip dengan aplikasi windows kebanyakan. Begitupun juga dengan Linux dan Mac. Pendekatan ini berbeda dengan Java dengan pustaka Swing-nya.

Di Linux, sistem GUI wx diimplementasikan dengan berbagai pustaka GUI lain. Kita mengenal implementasi wx untuk X11 (wxX11), wx untuk Motif (wxMotif), wx untuk GTK (wxGTK, implementasi yang sangat matang, kita akan mempergunakan implementasi yang satu ini), wxNanox, dan lain sebagainya.

Di Windows, sistem GUI wx diimplementasikan menjadi wxMSW (wx Microsoft Windows). Di Mac, sistem GUI wx diimplementasikan menjadi wxMac. Semua implementasi sebisa mungkin mempergunakan pustaka level bawah yang kompatibel dengan sistem.

Berikut ini adalah kesimpulan keuntungan yang didapatkan ketika menggunakan wx.

1. Gratis.
2. Kita dapat memiliki source code wx.
3. tersedia dalam berbagai platform populer.
4. Bekerja dengan sebagian besar kompilator C++ dan Python (binding untuk Python sangatlah matang).
5. Datang dengan banyak contoh (beberapa contoh memang tidak dapat bekerja atau sudah ketinggalan zaman, namun tetap merupakan dasar yang baik sebagai contoh).
6. Dokumentasi yang sangat lengkap (lebih dari 1500 halaman, siap cetak). Dokumentasi untuk setiap kelas sangatlah jelas dan lengkap.
7. Datang bersama tool Tex2RTF yang dapat digunakan untuk membangun sistem help canggih.
8. Mudah digunakan dan berorientasi objek
9. Sistem event yang fleksibel. Yang satu ini memang perlu dipuji. Ketika menggu-

nakan Delphi misalnya, dengan mudah kita bisa mengasosiasikan event tertentu yang dikenakan pada suatu objek dengan fungsi *event handling* tertentu. Namun, perkembangan sistem event di Delphi juga terus mengalami perkembangan. Event handling bukan isu yang sederhana di dalam pemrograman. Di penerapan lain seperti GTK+, event handling benar-benar bukan masalah yang sederhana. Dengan penggunaan event table di wx, event handling menjadi jauh lebih mudah untuk dilakukan.

10. Dapat digunakan untuk membuat aplikasi yang kayak grafik.
11. Sistem *layout* yang canggih. Pada kenyataannya, tersedia banyak sistem layout yang ditawarkan. Mulai dari sistem kuno dengan sistem peletakan absolut, ataupun dengan sistem sizer yang fleksibel.
12. Arsitektur Print/Preview dan Document/View.
13. Tersedia berbagai widget modern seperti tree control, notebook, toolbar (yang mendukung docking dan floating), grid, dan lain sebagainya.
14. Dukungan MDI.
15. Dapat digunakan untuk membangun DLL di Windows, atau shared object di Linux.
16. Tersedia berbagai dialog siap pakai seperti File dialog, color dialog, print dialog dan lain sebagainya. Dan, yang jelas, cukup mudah digunakan.
17. Dukungan clipboard di Windows.
18. Dukungan sistem help yang canggih, serta tersedia API untuk mengakses sistem help.
19. Disertai dengan HTML engine. Tidak canggih-canggih sekali, namun cukup untuk digunakan.
20. Dilengkapi dengan dialog editor (tersedia banyak sekali, mulai dari yang komersial sampai yang gratis) untuk membantu membangun sistem GUI berbasis XML. Fitur yang satu ini membuat penulis jatuh cinta setengah mati. Penggunaannya pun tidak sangat susah sekali.
21. Dukungan pustaka jaringan.
22. Mendukung pemrosesan image yang platform independen.
23. Mendukung berbagai format gambar populer seperti BMP, PNG, JPEG, GIF,

XPM, PNM dan PCX.

24. Datang dengan pustaka threading yang kuat.

Harus diakui, membangun aplikasi GUI menggunakan C++ dan Python, walaupun sama-sama menggunakan pustaka Wx, jauh lebih cepat, mudah dan bersih menggunakan Python. Namun, sekali lagi, ada yang harus dibayar.

Di dalam artikel ini, setelah membahas hal-hal nonteknis, kita akan memasuki pembahasan-pembahasan mendasar pemrograman GUI menggunakan wx. Mulai dari contoh yang dasar sekali, pembuatan makefile yang sederhana, sampai aplikasi yang lebih modern.

Bagi Anda yang tidak menggunakan C++, setiap contoh akan dibahas dari pemahaman yang sangat mendasar. Tentunya, Anda perlu menguasai sintaks-sintaks dasar C dan C++ seperti deklarasi kelas, perulangan, kondisi, pemahaman *inheritance* dan lain sebagainya.

Paket yang dibutuhkan

Umumnya, distro-distro desktop populer telah membungkus wx dengan widget set GTK (wxGTK) di dalam distronya (biasanya dengan nama wxGTK-xyz.rpm untuk distro berbasis RPM). Apabila Anda bisa menjalankan audacity atau xchm misalnya, maka sudah pasti runtime wxGTK telah terinstal di sistem.

Namun, untuk membangun aplikasi, memiliki wxGTK terinstall saja tidak cukup. Anda akan membutuhkan header-header wxGTK yang umumnya dipaketkan terpisah dalam paket wxGTK-devel.

Hello World

Kita akan mulai dengan contoh yang sangat sederhana, sebuah aplikasi yang datang dengan form kosong dengan title bar bertuliskan Hello World. Title bar akan mengandung pula button-button kontrol window manager yang berfungsi. Sebagai tambahan, kita juga akan membuat status bar dengan dua bagian, di mana bagian pertama juga akan bertuliskan Hello World.

Sebutlah aplikasi kita dengan nama helloworld. Di dalam pengembangan aplikasi kita ini, kita akan melibatkan sebuah file helloworld.cpp dan sebuah makefile dengan

nama makefile.unx. Untuk saat ini, semua deklarasi kelas dan kode-kode disimpan dalam helloworld.cpp.

Dalam membangun sebuah aplikasi wx, kita perlu menurunkan kelas utama aplikasi kita dari kelas wxApp. Aplikasi kita, tidak perlu memiliki fungsi main() seperti aplikasi C biasa. Namun, kita perlu meng-override fungsi OnInit() milik kelas utama aplikasi kita. Di dalam OnInit() inilah, kita membuat window, dan lain sebagainya.

Aplikasi kita juga perlu menggunakan header <wx/wx.h>. Header ini telah menginclude sebagian besar header lain yang umum diperlukan. Dalam kebanyakan aplikasi, menggunakan header wx.h saja sudah cukup.

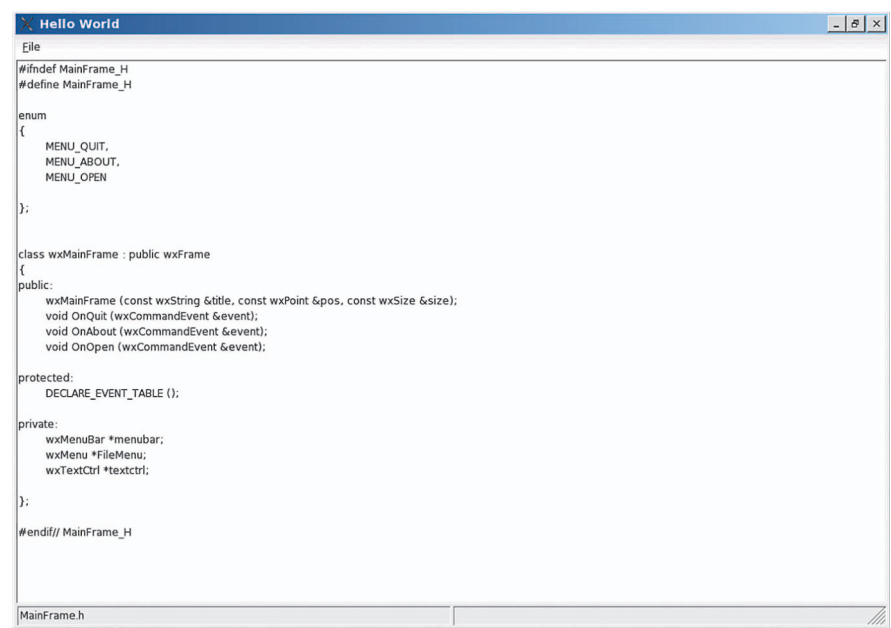
Selain kelas utama aplikasi, umumnya kita akan memiliki pula kelas untuk penanganan window utama aplikasi. Di sinilah kita menambahkan berbagai kontrol lain dan memroses event yang dikenakan pada setiap komponen window utama.

Berikut ini adalah keseluruhan isi dari helloworld.cpp. Penjelasan akan dibahas setelahnya.

```
#include <wx/wx.h>
class wxHelloWorld : public
wxApp
{
public:
    virtual bool OnInit ();
};
```

```
DECLARE_APP (wxHelloWorld)

class wxMainFrame : public
wxFrame
{
public:
    wxMainFrame (const wxString
&title, const wxPoint &pos,
const wxSize &size);
};
IMPLEMENT_APP (wxHelloWorld)
bool wxHelloWorld :: OnInit ()
{
    wxMainFrame *frame = new
wxMainFrame ("Hello World",
wxPoint (50,50), wxSize
(200,200));
    SetTopWindow (frame);
    frame -> Show (TRUE);
    return TRUE;
};
wxMainFrame :: wxMainFrame
(const wxString &title, const
wxPoint &pos, const wxSize
&size):
    wxFrame ( (wxFrame *)
NULL, -1, title, pos, size )
{
    CreateStatusBar (2);
    SetStatusText ("Hello
World");
};
```



Hello World Application.

Kode tersebut sepenuhnya berfungsi. Berikut ini adalah penjelasan bagian-bagian dalam helloworld.cpp:

- Setiap aplikasi kita perlu menggunakan header wx.h.
- Pertama-tama, seperti yang telah disebutkan sebelumnya, aplikasi kita akan memiliki sebuah kelas utama yang diturunkan (secara public) dari kelas wx-App.

```
class wxHelloWorld : public
wxApp
{
public:
    virtual bool OnInit ();
};
```

Kita sebut nama kelas utama kita dengan wxHelloWorld. Kelas ini akan meng-override fungsi OnInit (), yang akan dibahas kemudian.

- Untuk window utama, kita perlu pula menurunkan sebuah kelas dari wx-Frame.

```
class wxMainFrame : public
wxFrame
{
public:
    wxMainFrame (const
wxString &title, const
wxPoint &pos, const wxSize
&size);
};
```

Kita sebut nama kelas window utama kita sebagai wxMainFrame. Untuk saat ini, apa yang kita miliki hanyalah *constructor* kelas kita. Di constructor inilah kita akan membuat status bar dan lain sebagainya.

- Di dalam fungsi OnInit () kelas utama kita, wxHelloWorld :: OnInit (), kita membuat instance baru dari wxMainFrame yang kita miliki, memberikan title Hello World, mengubah ukuran, menampilkannya, serta meminta instance wxMainFrame kita sebagai window utama. Fungsi OnInit () ini mengembalikan nilai boolean. Apabila kode-kode di dalam fungsi ini gagal dikerjakan, maka aplikasi akan keluar.

```
bool wxHelloWorld :: OnInit ()
{
    wxMainFrame *frame
= new wxMainFrame ("Hello
```

```
World", wxPoint (50,50),
wxSize (200,200));
    SetTopWindow (frame);
    frame -> Show (TRUE);
    return TRUE;
};
```

- Sementara, di dalam constructor wxMainFrame kita, kita membuat sebuah status bar dengan dua bagian, dan menu-liskan Hello World pada bagian pertama.

```
wxMainFrame :: wxMainFrame
(const wxString &title, const
wxPoint &pos, const wxSize
&size):
    wxFrame ( (wxFrame
*) NULL, -1, title, pos,
size )
{
    CreateStatusBar (2);
    SetStatusText ("Hello
World");
};
```

Harap diperhatikan bahwa wxMainFrame dalam deklarasinya diturunkan dari wxFrame.

- Aplikasi kita tidak memiliki fungsi main(), namun dengan bantuan macro seperti DECLARE_APP (wxHelloWorld) dan IMPLEMENT_APP (wxHelloWorld), apa yang dibutuhkan untuk menjadikan HelloWorld sebagai aplikasi telah dilakukan untuk kita.

Apabila diperhatikan, kode kita cukup sederhana dan bisa dimengerti. Kali pertama penulis mempelajari wx, kesan yang didapat adalah kode wx sangatlah bersih dan mudah dimengerti, terutama dibandingkan dengan GTK+ secara native.

Setelah helloworld.cpp, kita perlu membuat makefile agar kompilasi dapat dilakukan dengan mudah. Kita akan berikan nama makefile.unx untuk platform UNIX, dan makefile.msw untuk platform Microsoft. Untuk saat ini, kita hanya akan membahas makefile.unx. Berikut ini adalah keseluruhan isi makefile.unx:

```
CXX = $(shell wx-config --cxx)
PROGRAM = helloworld
OBJECTS = helloworld.o
# implementation
.SUFFIXES: .o .cpp
```

```
.cpp.o :
    $(CXX) -c `wx-config --
cxxflags` -o $@ $<
all: $(PROGRAM)
$(PROGRAM): $(OBJECTS)
    $(CXX) -o $(PROGRAM)
$(OBJECTS) `wx-config --libs`
clean:
    rm -f *.o $(PROGRAM)
```

Makefile ini banyak mempergunakan program wx-config untuk membantu mendapatkan flag C++ dan library. Makefile kita ini memiliki beberapa tugas, yaitu make all untuk menghasilkan program (helloworld) dan make clean untuk membersihkan program dan sampah kompilasi.

Makefile ini bisa digunakan untuk aplikasi kita yang lain. Apa yang perlu diubah umumnya hanya variabel PROGRAM dan OBJECTS.

Setelah memiliki makefile, kita dapat mengompilasi helloworld.cpp dengan perintah berikut ini:

```
$ make -f makefile.unx
g++ -c `wx-config --cxxflags` -o
helloworld.o helloworld.cpp
g++ -o helloworld helloworld.o
`wx-config -libs`
```

Sekarang, Anda akan memiliki dua file tambahan: helloworld (aplikasi kita) dan helloworld.o (objek hasil kompilasi). Perhatikanlah ukuran helloworld kita. Di sistem penulis, ukurannya sangatlah kecil, hanya sekitar 40 KB.

Cobalah jalankan aplikasi kita. Sebuah form dengan status bar dan title bar bertuliskan Hello World akan ditampilkan. Tombol-tombol pada title bar telah berfungsi sepenuhnya. Hello World kita telah selesai. Aplikasi kita nantinya bisa diporting ke platform lain dengan modifikasi minimal (modifikasi minimal diperlukan karena kita tidak memberikan penanda tertentu untuk aplikasi yang siap langsung dikompilasi di platform lain).

Bagaimana kesan Anda tentang wx setelah aplikasi minimal ini? Kode yang kita bahas belumlah siap digunakan. Di edisi mendatang, kita akan membahas cara pembuatan kode yang lebih baik. Sampai bulan depan!

Noprianto (noprianto@infolinux.co.id)

Pemrograman Multiplatform dengan WxWindows

Bagian 2 dari 2 Artikel

C++ adalah salah satu bahasa berorientasi objek yang matang dan telah digunakan dalam pembangunan banyak aplikasi skala besar. Dengan digabungkan dengan wx, kita bisa membangun aplikasi berbasis GUI yang canggih, dengan biaya yang tidak terlalu besar. Di artikel lanjutan ini, kita akan membahas cara pemrograman wx yang lebih baik lagi.

Ketika penulis beberapa lama berke-nalan dengan *open source*, penulis merasa ingin melihat source code yang terbuka tersebut. Ingin mempelajari. Penulis memulai dengan aplikasi kecil yang dibuat dengan bahasa C/C++.

Dari hasil membuka-buka source tersebut, penulis ternyata cukup terkejut karena beberapa hal berikut. Yang pertama. Rupanya, untuk membuat aplikasi yang kecil sekalipun, banyak sekali file source C/C++ yang beberapa di antaranya membutuhkan file lain. Sementara, ada satu atau dua file yang hanya berisikan beberapa baris kode. Hal ini bertentangan dengan pemahaman penulis tentang pembuatan program. Waktu itu, penulis berpendapat bahwa untuk membuat sebuah program dengan fungsi spesifik, mungkin file source yang dibutuhkan hanya satu atau dua, dengan sebagian besar fungsi/logika disimpan dalam satu atau dua file tersebut. Waktu itu, penulis benar-benar tidak mengerti alasan programmer aplikasi-aplikasi tersebut untuk memisahkan kelas, kode, header dan lain sebagainya. Butuh waktu cukup lama untuk mengerti. Yang kedua, yang cukup membuat penulis sadar, rupanya file header sebisa mungkin tidak diisi dengan logika program.

Kita telah membahas pemrograman C++ dan Wx di edisi yang lalu. Kita juga tidak melakukan separasi kelas, kode program dan header sehingga semua menumpuk dalam satu atau dua file. Di edisi ini, kita akan menulis ulang aplikasi kemarin menjadi lebih baik, lebih siap dikembangkan.

Separasi kelas, kode, dan header

Walaupun tidak wajib, sebaiknya kita memberikan separasi yang jelas antara setiap kelas dan definisi kelas serta kode-kode pembentuk kelas. Berikut ini, kita akan menuliskan aplikasi HelloWorld kita sebelumnya menjadi lebih bagus lagi. Hasilnya sama, tetap aplikasi Hello World.

Pertama-tama, karena kita memiliki dua kelas (wxHelloWorld dan wxMainFrame), maka kita akan memiliki pula dua file untuk definisi kerangka kelas, sebuahlah dengan nama helloworld.h dan MainFrame.h

Yang kedua, karena setiap header helloworld.h dan MainFrame.h hanya mendefinisikan kelas, kita perlu file pendukung untuk kode kelas wxHelloWorld dan wxMainFrame. Sebutlah dengan nama helloworld.cpp dan MainFrame.cpp.

Dengan demikian, dari satu file (helloworld.cpp), program kita akan berkembang menjadi empat file (helloworld.h, MainFrame.h, helloworld.cpp dan MainFrame.cpp). Berikut ini adalah isi dari masing-masing file.

helloworld.h (di header ini, kita hanya mendeklarasikan kelas wxHelloWorld):

```
#ifndef HelloWorld_H
#define HelloWorld_H

class wxHelloWorld : public
wxApp
{
public:
    virtual bool OnInit ();
};
```

```
DECLARE_APP (wxHelloWorld)

#endif // HelloWorld_H
```

MainFrame.h (di header ini, kita hanya mendeklarasikan kelas wxMainFrame):

```
#ifndef MainFrame_H
#define MainFrame_H

class wxMainFrame : public
wxFrame
{
public:
    wxMainFrame (const
wxString &title, const wxPoint
&pos, const wxSize &size);
};

#endif // MainFrame_H
```

MainFrame.cpp (di file ini, kita menuliskan kode-kode kelas wxMainFrame. Harap diperhatikan, kita perlu menggunakan header MainFrame.h):

```
#include <wx/wx.h>
#include "MainFrame.h"

wxMainFrame :: wxMainFrame
(const wxString &title, const
wxPoint &pos, const wxSize
&size):
    wxFrame ( (wxFrame *)
NULL, -1, title, pos, size )
{
```

```
CreateStatusBar (2);
SetStatusText ("Hello
World");
};
```

helloworld.cpp (di file ini, kita menuliskan kode-kode kelas wxHelloWorld. Harap diperhatikan, kita perlu menggunakan header helloworld.h dan MainFrame.h):

```
#include <wx/wx.h>
#include "helloworld.h"
#include "MainFrame.h"

IMPLEMENT_APP (wxHelloWorld)

bool wxHelloWorld :: OnInit ()
{
    wxMainFrame *frame = new
    wxMainFrame ("Hello World",
    wxPoint (50,50), wxSize
    (200,200));

    SetTopWindow (frame);

    frame -> Show (TRUE);

    return TRUE;
};
```

Kini, kita telah menuliskan kerangka kelas dan kode dengan benar. Setiap kelas satu file, dan separasi dilakukan antara definisi kelas dan kode. Namun sayangnya, makefile.unx kita menjadi ketinggalan zaman dengan pengubahan ini. Kenapa? Pada variabel OBJECTS, kita hanya mendeklarasikan objek helloworld.o. Kita perlu melakukan sedikit modifikasi agar makefile kita bisa digunakan. Berikut ini adalah makefile.unx yang baru:

```
CXX = $(shell wx-config --cxx)

PROGRAM = helloworld

OBJECTS = helloworld.o
MainFrame.o

# implementation

.SUFFIXES:      .o .cpp
```

```
.cpp.o :
    $(CXX) -c `wx-config --
cxxflags` -o $@ $<

all:      $(PROGRAM)

$(PROGRAM):      $(OBJECTS)
    $(CXX) -o $(PROGRAM)
$(OBJECTS) `wx-config --libs`

clean:
    rm -f *.o $(PROGRAM)
```

Satu-satunya perubahan adalah dengan menambahkan MainFrame.o pada variabel OBJECTS. Kompilasi program kita kemudian dapat dilakukan dengan perintah yang sama.

```
$ make -f makefile.unx
g++ -c `wx-config --cxxflags` -o
helloworld.o helloworld.cpp
g++ -c `wx-config --cxxflags` -o
MainFrame.o MainFrame.cpp
g++ -o helloworld helloworld.o
MainFrame.o `wx-config -libs`
```

Kompilasi kita bertambah satu baris, yaitu dengan kompilasi MainFrame.o secara terpisah. Kini, aplikasi kita siap dijalankan.

Menambahkan kontrol lain dan menangani event

Kini, aplikasi kita telah dapat menampilkan sebuah form kosong. Kita akan melengkapi dengan menu bar, text control dan contoh penggunaan dialog untuk membuka file dan menampilkannya ke text control tersebut. Aplikasi kita juga akan memiliki sebuah *about box* yang sangat sederhana.

Semua perubahan ini akan dikerjakan pada kelas wxMainFrame. Oleh karena itu, file yang harus diedit adalah MainFrame.h dan MainFrame.cpp.

Karena menu bar dan text control adalah milik dari wxMainFrame, maka kita akan mendeklarasikan pada bagian *private* kelas wxMainFrame. Berikut ini adalah file MainFrame.h yang baru:

```
#ifndef MainFrame_H
#define MainFrame_H

enum
{
```

```
MENU_QUIT,
MENU_ABOUT,
MENU_OPEN

};

class wxMainFrame : public
wxFrame
{
public:
    wxMainFrame (const
wxString &title, const wxPoint
&pos, const wxSize &size);

private:
    wxMenuBar *menubar;
    wxMenu *FileMenu;
    wxTextCtrl *textctrl;

};

#endif// MainFrame_H
```

File ini belum sepenuhnya final karena belum melibatkan penanganan event. Namun, sampai langkah ini, kita telah memiliki tiga anggota *private* kelas yaitu *menubar*, *FileMenu*, dan *textctrl*.

Berikut ini adalah file MainFrame.cpp yang baru (juga belum sepenuhnya final karena belum melibatkan penanganan event).

```
#include <wx/wx.h>
#include "MainFrame.h"

wxMainFrame :: wxMainFrame
(const wxString &title, const
wxPoint &pos, const wxSize
&size):
    wxFrame ( (wxFrame *)
NULL, -1, title, pos, size )
{
    CreateStatusBar (2);
    SetStatusText ("Hello
World");

    menubar = new wxMenuBar;

    FileMenu = new wxMenu;
    FileMenu -> Append (MENU_
ABOUT, "&About", "Show about
info...");
    FileMenu -> AppendSeparator
```

```
();

FileMenu -> Append (MENU_
OPEN, "&Open", "Open file...");
FileMenu -> AppendSeparator
();

FileMenu -> Append (MENU_
QUIT, "&Quit", "Quit from
application...");

menubar -> Append
(FileMenu, "&File");

SetMenuBar (menubar);

textctrl = new wxTextCtrl
(this, -1, wxString
("Halo"), wxDefaultPosition,
wxDefaultSize, wxTE_MULTILINE);

};
```

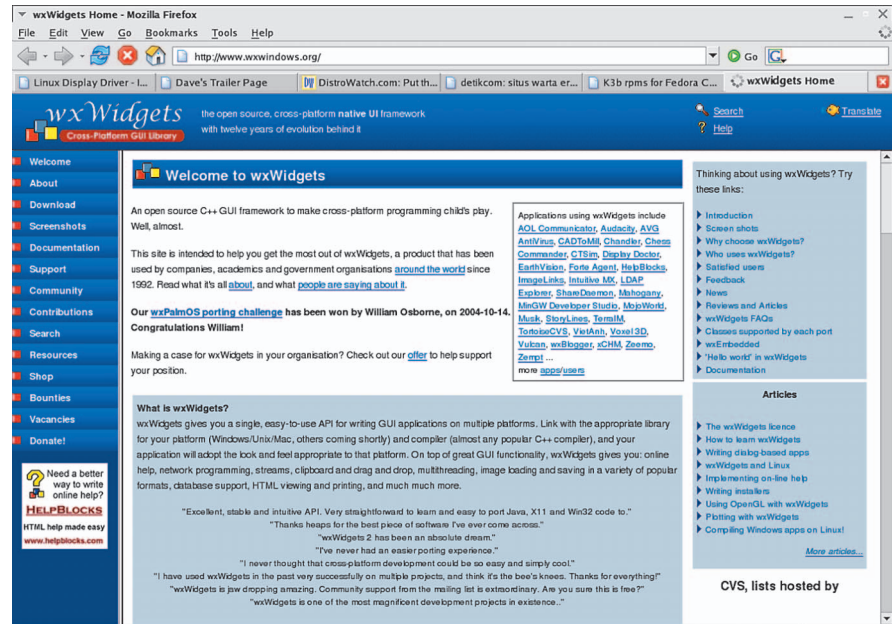
Apabila dikompilasi ulang, maka kini Anda akan memiliki helloworld yang baru. Helloworld yang ketika dijalankan akan memiliki menu bar dan text control. Menu bar juga akan mengandung menu File dengan submenu *About*, *Open*, dan *Quit*.

Sampai saat ini, apa yang kita miliki hanyalah pengenalan untuk submenu About, Open, dan Quit. Kita belum memiliki *event handling* sama sekali. Katakanlah, untuk menangani setiap submenu, kita akan memiliki fungsi sendiri. Sebut saja dengan nama OnAbout, OnOpen, dan OnQuit. Kita perlu mengubah MainFrame.h untuk menambahkan deklarasi fungsi dan deklarasi tabel event. Berikut ini adalah isi MainFrame.h yang baru (dan final):

```
#ifndef MainFrame_H
#define MainFrame_H

enum
{
    MENU_QUIT,
    MENU_ABOUT,
    MENU_OPEN
};

class wxMainFrame : public
wxFrame
```



Situs web wxWindows

```
{
public:
    wxMainFrame (const
wxString &title, const wxPoint
&pos, const wxSize &size);
    void OnQuit (wxCommandEvent
&event);
    void OnAbout
(wxCommandEvent &event);
    void OnOpen (wxCommandEvent
&event);

protected:
    DECLARE_EVENT_TABLE ();

private:
    wxMenuBar *menubar;
    wxMenu *FileMenu;
    wxTextCtrl *textctrl;
};

#endif// MainFrame_H
```

Kita menambahkan `DECLARE_EVENT_TABLE()` untuk mendeklarasikan event tabel untuk setiap event yang akan diproses. Berikut ini adalah isi file MainFrame.cpp yang baru (dan final).

```
#include <wx/wx.h>
#include "MainFrame.h"

BEGIN_EVENT_TABLE (wxMainFrame,
```

```
wxFrame)

    EVT_MENU (MENU_QUIT,
wxMainFrame :: OnQuit)
    EVT_MENU (MENU_ABOUT,
wxMainFrame :: OnAbout)
    EVT_MENU (MENU_OPEN,
wxMainFrame :: OnOpen)
END_EVENT_TABLE ()

wxMainFrame :: wxMainFrame
(const wxString &title, const
wxPoint &pos, const wxSize
&size):
    wxFrame ( (wxFrame *)
NULL, -1, title, pos, size )
{
    CreateStatusBar (2);
    SetStatusText ("Hello
World");

    menubar = new wxMenuBar;

    FileMenu = new wxMenu;
    FileMenu -> Append (MENU_
ABOUT, "&About", "Show about
info...");
    FileMenu -> AppendSeparator
();
    FileMenu -> Append (MENU_
OPEN, "&Open", "Open file...");
    FileMenu -> AppendSeparator
```

```

());
FileMenu -> Append
(MENU_QUIT, "&Quit", "Quit from
application...");

menubar -> Append
(FileMenu, "&File");

SetMenuBar (menubar);

textctrl = new wxTextCtrl
(this, -1, wxString ("Halo"),
wxDefaultPosition,
wxDefaultSize, wxTE_MULTILINE);

Connect (MENU_QUIT,
wxEVT_COMMAND_MENU_SELECTED,
(wxObjectEventFunction)
&wxMainFrame :: OnQuit);
Connect (MENU_ABOUT,
wxEVT_COMMAND_MENU_SELECTED,
(wxObjectEventFunction)
&wxMainFrame :: OnAbout);
Connect (MENU_OPEN,
wxEVT_COMMAND_MENU_SELECTED,
(wxObjectEventFunction)
&wxMainFrame :: OnOpen);

};

void wxMainFrame :: OnQuit
(wxCommandEvent &event)
{
    Close (TRUE);
};

void wxMainFrame :: OnAbout
(wxCommandEvent &event)
{
    wxMessageBox ("Hello
World Example v0.1", "About
Hello World", wxOK | wxICON_
INFORMATION, this);
};

void wxMainFrame :: OnOpen
(wxCommandEvent &event)
{
    wxFileDialog *dlg = new
wxFileDialog (this, "open a text

```

```

file", "", "",
    "All Files
(*.*)|*.|*.txt files
(*.txt)|*.txt", wxOPEN,
wxDefaultPosition);

    if (dlg -> ShowModal () ==
wxID_OK)
    {
        textctrl -> LoadFile
(dlg -> GetFilename () );

        SetStatusText (dlg
-> GetFilename (), 0);

    };

    dlg -> Destroy ();
};

```

Kita menambahkan cukup banyak kode ke dalam MainFrame.cpp kita. Berikut ini adalah penjelasannya:

Pemberian event yang telah dideklarasikan secara protected pada definisi kelas wxMainFrame dilakukan dengan cara pembuatan tabel event:

```

BEGIN_EVENT_TABLE (wxMainFrame,
wxFrame)

    EVT_MENU (MENU_QUIT,
wxMainFrame :: OnQuit)
    EVT_MENU (MENU_ABOUT,
wxMainFrame :: OnAbout)
    EVT_MENU (MENU_OPEN,
wxMainFrame :: OnOpen)
END_EVENT_TABLE ()

```

Tabel event kita untuk saat ini hanya menangani EVT_MENU (), yaitu penanganan event dari menu. Setiap deklarasi isi tabel menggunakan ID yang telah dideklarasikan di dalam MainFrame.h. Setiap ID akan dihubungkan dengan fungsi yang telah disiapkan.

Ada satu hal yang bisa kita pelajari di sini. Borland, sekitar tahun 1996 pernah mengumumkan bahwa Delphi memiliki shared event handling. Satu event handler yang bisa dipakai bersama-sama.

Dengan wx, hal tersebut dapat dilakukan dengan mudah. Cukup berikan ID yang sama, atau handler yang sama, maka shared event handling bisa diimplementasikan.

Setelah dideklarasikan pada tabel event, kita secara eksplisit menghubungkan ID tertentu dengan fungsi yang telah dipersiapkan di constructor wxMainFrame.

```

Connect (MENU_QUIT,
wxEVT_COMMAND_MENU_SELECTED,
(wxObjectEventFunction)
&wxMainFrame :: OnQuit);
Connect (MENU_ABOUT,
wxEVT_COMMAND_MENU_SELECTED,
(wxObjectEventFunction)
&wxMainFrame :: OnAbout);
Connect (MENU_OPEN,
wxEVT_COMMAND_MENU_SELECTED,
(wxObjectEventFunction)
&wxMainFrame :: OnOpen);


```

Fungsi wxMainFrame :: OnAbout () dan wxMainFrame :: OnQuit () kemudian dipersiapkan dengan isi masing-masing adalah penampilan *message box* dan keluar dari *mainframe*. Keluar dari *mainframe* yang merupakan frame utama berarti keluar dari aplikasi karena di wxHelloWorld :: OnInit (), kita tidak memiliki penanganan apa-apa lagi.

Pembuatan dialog dapat dilakukan dengan sangat mudah. Yang perlu diperhatikan adalah bahwa kita membuat dialog secara dinamis dan menghancurkannya pula dengan pemanggilan method Destroy (). Membuka dan melakukan *loading* ke dalam text control dapat dilakukan dengan sangat mudah, hanya dengan memanggil method LoadFile ().

Kini, lakukanlah kompilasi ulang. Dan, setelah itu, aplikasi kita telah siap digunakan. Cobalah untuk mengakses setiap menu itemnya.

Bisa kita lihat, pemrograman GUI menggunakan pustaka wx dicapai dengan menggunakan API yang bersih dan potensial untuk dimengerti. Perbedaan dengan pemrograman a la Delphi misalnya adalah, bahwa kita mengerti betul setiap langkah pembuatan aplikasi kita. Mulai dari deklarasi kelas, menurunkan kelas, membuat event table, menambahkan kontrol dan lain sebagainya.

Sampai di sini dulu perkenalan kita dengan wx. Kita akan terus melanjutkan dengan pembahasan wx lainnya. Selamat mencoba, dan sampai ketemu lagi! 
Noprianto (noprianto@infolinux.co.id)