Ariya Hidayat
Email: ariya@kde.org
URL: http://ariya.pandu.org

Tutorial Qt (bagian 1)

Seri tulisan ini mencoba menghantarkan topik seputar pemrograman aplikasi GUI di Linux dengan menggunakan Qt. Untuk itu, diasumsikan bahwa pembaca telah memahami C++, pemrograman berorientasi objek (OOP), dan tidak asing dengan GNU C/C++ Compiler untuk pengembangan aplikasi Unix. Sebagai gambaran kasar, mula-mula akan diulas dahulu mengenai penggunaan Qt dan kelak akan perlahan-lahan menuju KDE.

Membangun aplikasi berbasis GUI di Unix mengharuskan penggunaan Application Programming Interface (API) dari X Window. Bertahun-tahun yang silam, hanya API ini yang tersedia. Tidak heran bahwa API yang lengkap namun relatif ekstensif dan berat menyebabkan tidak semua programmer mempunyai cukup energi untuk mempelajarinya. Dus, mengembangkan aplikasi untuk X bukanlah hal yang menyenangkan bagi tiap orang.

Belakangan muncul *toolkit* yang memang dirancang untuk memudahkan pembuatan aplikasi dengan bertindak selaku lapis abstraksi, selain juga untuk memicu upaya konsistensi *look-and-feel* antara satu program dengan program lainnya. Dengan menggunakan toolkit, API X Window yang rumit menjadi disembunyikan melalui sekumpulan fungsi-fungsi yang lebih sederhana. Lebih jauh, setelah kehadiran toolkit, muncul pula *desktop environment* seperti CDE, KDE, dan GNOME yang meningkatkan derajat abstraksi pengembangan aplikasi menjadi lebih dari sekedar user interface semata.

Qt adalah toolkit yang digunakan untuk membangun aplikasi berbasis GUI di Unix. Qt dikembangkan oleh Trolltech [1] dan menjadi fondasi untuk K Desktop Environment (KDE), selain juga telah dipergunakan dalam berbagai aplikasi komersial. Qt digolongkan sebagai software open-source karena dilisensi dual: dengan General Public License (GPL) dan Qt Public License (QPL). Khusus untuk tulisan ini, yang akan dibahas adalah Qt versi 2.x.

Qt dirancang untuk pengembangan aplikasi dengan C++. Oleh karenanya, Qt berisi sekumpulan kelas-kelas yang tinggal dimanfaatkan saja, mulai dari urusan antarmuka (user interface), operasi input ouput, networking, timer, template library, dan lain-lain. Qt mendukung penuh Unicode (mulai versi 2.0) sehingga urusan internationalization (I18N) dan encoding teks bukan menjadi masalah. Walaupun merupakan *free software*, Qt terbukti stabil dan lengkap. Dibandingkan toolkit lain, Qt juga mudah untuk dipelajari dan dipersenjatai dengan dokumentasi dan tutorial yang ekstensif dan rinci.

Untuk dapat mempelajari Qt, syarat utama adalah tersedia Qt pada sistem Anda. Bila tidak, maka Qt bisa didownload terlebih dahulu dari situs web Qt di Trolltech.com. Selanjutnya, Anda bisa mengikuti instruksi yang diberikan untuk melakukan compile dan

instalasi. Qt dirancang cocok untuk digunakan di lingkungan Unix, oleh karenanya Anda seharusnya tidak akan menemui hambatan dalam proses kompilasi.

Cara lain yang lebih menghemat waktu dan tenaga (dan juga bandwidth) adalah dengan mengambil Qt yang sudah dipaket dalam RPM atau DEB. Tentunya paket yang diambil harus sesuai dengan distribusi Linux yang digunakan. Lokasi paling gampang untuk memperoleh Qt versi RPM adalah pada server FTP untuk KDE 2, hal ini karena Qt 2.x selalu tersedia bersama KDE [2].

Hello World !

Sudah menjadi tradisi, bahwa untuk memulai mempelajari penggunaan sebuah bahasa pemrograman maupun library, contoh aplikasi pertama selalu yang berkenaan dengan "Hello World". Mengikuti kebiasaan ini, berikut adalah program berbasis Qt untuk menampilkan pesan yang sangat terkenal ini:

```
 1: // qhello.cpp - Qt Hello world
 2:
 3: #include <qapplication.h>
 4: #include <qlabel.h>
 5:
 6: int main( int argc, char **argv )
 7: {
 8:   QApplication app( argc, argv );
 9:   QString msg( "Hello world !" );
10:
11:   QLabel txt( "Hello world !", 0 );
12:   txt.resize( 250,50 );
13:   txt.setAlignment( Qt::AlignCenter );
14:
15:   app.setMainWidget( &txt );
16:   txt.show();
17:
18:   return app.exec();
19: }
```

Listing 1: qhello.cpp

Mengasumsikan Qt terinstal di `/usr/lib/qt2`, maka contoh program di atas dapat dikompilasi dengan perintah berikut ini:

```
g++ -o qhello qhello.cpp -lqt -I /usr/lib/qt2/include -L /usr/lib/qt2/lib
```

(Belakangan Makefile akan digunakan untuk memudahkan proses kompilasi sehingga tidak perlu melalui perintah yang panjang seperti di atas).

Pada perintah di atas, option `-I` menentukan lokasi pencarian file header yang akan di-include dalam program sementara option `-L` untuk memberi informasi lokasi library untuk proses link, dalam hal ini adalah `libqt.so`. Bisa dilihat, bahwa dengan option `-l`, contoh program ini dilink dengan library `qt`.

Bila tidak ada yang salah, maka program `qhello` akan dihasilkan dan siap dijalankan (dengan `./qhello`). Anda memperoleh hasil kira-kira seperti:

Gambar 1: Screenshot qhello

Tentu saja Anda mungkin mendapatkan hasil yang berbeda karena baik dekorasi window dan style dari program akan sangat tergantung kepada window manager yang digunakan. Walaupun demikian, secara fungsional, program `qhello` ini tidak akan berubah. Contoh di atas adalah `qhello` yang dijalankan pada KDE dengan theme dari SuSE. Berikut adalah `qhello` yang berjalan pada WindowMaker dan GNOME:

Gambar 2: Screenshot qhello pada WindowMaker

Gambar 3: Screenshot qhello pada GNOME

Untuk memberikan gambaran dengan jelas akan apa yang terjadi, berikut akan diuraikan penjelasan mengenai source code program qhello ini.

Dua kelas utama yang digunakan di sini adalah QApplication dan QLabel. Yang pertama adalah untuk membangun sebuah aplikasi generik, sedangkan yang kedua dibutuhkan sebagai elemen user interface untuk menampilkan *label*, yaitu sebuah teks dalam window. Kedua kelas ini terlebih digunakan dengan meng-include-kan file header yang bersesuaian, yaitu `qapplication.h` dan `qlabel.h`.

Objek `app` adalah dari kelas QApplication. Dapat dilihat di sini bahwa QApplication selalu diinisialisasi dengan memberikan seluruh argumen program. Lebih rinci mengenai hal ini bisa dirujuk ke dokumentasi Qt.

Objek `txt` adalah label yang berasal dari kelas QLabel. Objek ini dikonstruksi dengan memberikan string yang akan ditambilkan. Argumen kedua pada konstruktornya, yaitu 0, bermakna bahwa QLabel ini tidak diikatkan ke objek lainnya.

Fungsi `setMainWidget` diperlukan agar label yang barus saja dibuat dijadikan widget utama window aplikasi. Dalam terminologi Qt, *widget* berarti objek yang berhubungan dengan user interface. Label adalah hanya salah satu jenis widget. Menu, button, kotak dialog, dan lain-lain adalah juga widget. Walaupun tidak harus, biasanya sebuah aplikasi memiliki widget utama, yang dalam hal ini diset dengan `setMainWidget`. Perhatikan bahwa setelahnya fungsi `show` perlu dipanggil secara eksplisit. Tanpa ini, widget tidak akan tampak (alias *hidden*).

Signal dan Slot

Salah satu karakteristik dari Qt adalah penggunaan *signal* dan *slot*. Karena Qt dibangun dengan C++, maka baik signal maupun slot terintegrasi sebagai bagian dari kelas-kelas yang terdapat pada Qt. Lebih jauh, signal/slot menjadi bagian dari *object model* yang digunakan Qt. Object model ini didesain khusus untuk memudahkan pemrograman aplikasi yang berbasis GUI.

Sebuah signal menandakan bahwa sesuatu telah terjadi. Signal dibangkitkan manakala terjadi interaksi antara user dan program, seperti misalnya ketika user mengklik mouse atau mengetikkan sesuatu di keyboard. Signal juga bisa dipicu oleh hal-hal yang merupakan internal program, misalnya karena sebuah timer yang diaktifkan sebagai alarm.

Slot adalah fungsi yang berespon terhadap signal tertentu. Untuk dapat melakukan hal ini, sebelumnya slot harus dikoneksikan dengan signal yang dimaksud.

Contoh program berikut menggambarkan penggunaan signal dan slot yang sederhana. Coba Anda cermati listing pogramnya berikut ini:

```
 1: // qclick.cpp - Qt Click-to-quit example
 2:
 3: #include <qapplication.h>
 4: #include <qpushbutton.h>
 5:
 6: int main( int argc, char **argv )
 7: {
 8:   QApplication app( argc, argv );
 9:
10:   QPushButton button( "Click to quit !", 0 );
11:   button.resize(250,50);
12:
13:
14:   QObject::connect( &button, SIGNAL(clicked()),
15:     &app, SLOT(quit()) );
16:
17:   app.setMainWidget( &button );
18:   button.show();
19:
20:   return app.exec();
21: }
```

Listing 2: qclick.cpp

Anda dapat mencoba dahulu untuk menjalankan program `qclick`. Lakukan compile dengan perintah berikut ini:

```
g++ -o qclick qclick.cpp -lqt -I /usr/lib/qt2/include -L /usr/lib/qt2/lib
```

Tampilan program jika dieksekusi kira-kira seperti berikut:

Gambar 4: Screenshot qclick

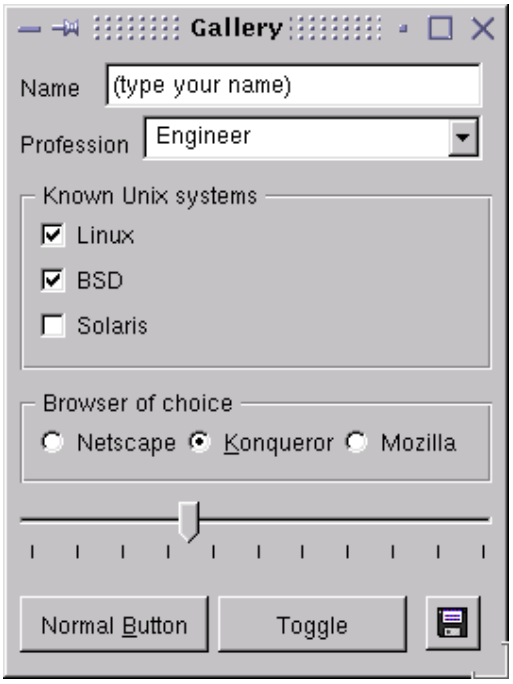
Program `qclick` tidak banyak berbeda dengan contoh sebelumnya. Di sini dimanfaatkan QPushButton untuk menghasilkan sebuah button pada window aplikasi, karenanya digunakan file header `qpushbutton.h`. Objek `button` dikonstruksi dengan memberikan string yang menjadi caption dari button tersebut.

Berbeda dengan contoh sebelumnya, program `qclick` ini sudah dapat sedikit berespons. Bila tombol "Click to quit" diklik, maka program akan segera diakhiri. Dapat dilihat pada source codenya, bahwa hal ini terjadi karena signal `QPushButton::clicked()` dari objek `button` dihubungkan dengan slot `QApplication::quit()` dari object `app`. Dalam dokumentasi Qt disebutkan bahwa QPushButton akan memicu (*emit*) signal `clicked()` kalau user mengklik tombol tersebut. Sementara itu, slot QApplication bernama `quit()`, manakala dieksekusi akan menyebabkan program utama berakhir.

Jelas bahwa fungsi `QObject::connect()` akan menyambungkan satu signal dengan sebuah slot tertentu. Baik signal maupun slot dapat saja dimiliki oleh objek yang berbeda. Pada contoh di atas, signal `QPushButton::clicked()` dihubungkan dengan slot `QApplication::quit()`.

Pagelaran Widget

User-interface yang hanya dibangun dari tombol (alias *button*) semata tentu saja tidak menarik. Pada ulasan di bawah, akan ditunjukkan beberapa widget yang paling sering digunakan untuk sebuah program aplikasi yang umum. Beberapa di antaranya adalah *label*, *edit box*, *list box*, *check box*, *radio button*, dan *slider*. Mari perhatikan sama-sama screenshot dari program ilustratif `Gallery`:



Gambar 5: Screenshot Program Gallery

Untuk menghasilkan program Gallery ini, dibutuhkan beberapa file yaitu `gallery.cpp`, `main.cpp`, dan sebuah `Makefile`.

Bagaimana widget-widget tersebut dirangkaikan menjadi satu ? Lihat file header `gallery.h` dan file `gallery.cpp` di bawah ini:

```
 1: // gallery.h
 2:
 3: #ifndef __GALLERY_H
 4: #define __GALLERY_H
 5:
 6: #include <qwidget.h>
 7:
 8: class Gallery: public QWidget
 9: {
10:    Q_OBJECT
11:
12:  public:
13:    Gallery();
14:
15: private:
16: };
17:
18: #endif
```

Listing 3: gallery.h

Perhatikan bahwa pada pendefinisiannya, digunakan `Q_OBJECT` yang berarti kelas ini menjadi bagian dari *object model* yang digunakan Qt. Dengan demikian, file header ini harus diproses dengan Meta Object Compiler (moc).

```
 1: // gallery.cpp - Gallery of some Qt widgets
 2:
 3: #include "gallery.moc"
 4:
 5: #include <qpushbutton.h>
 6: #include <qbuttongroup.h>
 7: #include <qcheckbox.h>
 8: #include <qradiobutton.h>
 9: #include <qlabel.h>
10: #include <qlineedit.h>
11: #include <qslider.h>
12: #include <qcombobox.h>
13:
14: // this is pixmap for a button
15: const static char * floppy_xpm[] = {
16: "16 16 11 1",
17: "  c None",
18: ". c #5D5D5D",
19: "+ c #000000",
20: "@ c #303030",
21: "# c #FFFFFF",
22: "$ c #800080",
23: "% c #DCDCDC",
24: "& c #C3C3C3",
25: "* c #000080",
26: "= c #0000FF",
27: "- c #585858",
28: "++++++++++++++++",
29: "+@+$=$*$=$*$=+@+",
30: "+#+############+",
31: "+@+##########+@+",
32: "+@+#$*$=$*$=#+@+",
33: "+@+##########+@+",
34: "+@+#$*$=$*$*#+@+",
35: "+@+##########+@+",
36: "+@@++++++++++@@+",
37: "+@@@@@@@@@@@@@@@@+",
38: "+@@@++++++++@@@+",
39: "+@@+#%#%%%&%+@@+",
40: "+@@+%@-&%&%&+@@+",
41: "+@@+#@-%&%&&+@@+",
42: "++@+%&%&%&&&+@@+",
43: " ++++++++++++++++"};
44:
45:
46: Gallery::Gallery(): QWidget( 0, "Widget" )
47: {
48:   setCaption("Gallery of Widgets");
49:   resize( 260, 330 );
50:
51:   // label
52:   QLabel* name_label = new QLabel ( "Name", this );
53:   name_label->move ( 5, 5 );
54:
```

```
 55:   // edit box
 56:   QLineEdit* name_editbox = new QLineEdit ( "(type your name)", this
);
 57:   name_editbox->move ( 50, 7 );
 58:   name_editbox->resize ( 200, 24 );
 59:
 60:   // another label
 61:   QLabel* prof_label = new QLabel ( "Profession", this );
 62:   prof_label->move ( 5, 35 );
 63:
 64:   // combo box
 65:   QComboBox* prof = new QComboBox ( this );
 66:   prof->insertItem ( "Engineer" );
 67:   prof->insertItem ( "Student" );
 68:   prof->insertItem ( "Professor" );
 69:   prof->resize ( 180, 24 );
 70:   prof->move ( 70, 35 );
 71:
 72:   // some check boxes
 73:   QButtonGroup* bg = new QButtonGroup ( 3, Qt::Vertical, "Known Unix
systems", this );
 74:   QCheckBox* linux_check = new QCheckBox ( "Linux", bg );
 75:   QCheckBox* bsd_check = new QCheckBox ( "BSD", bg );
 76:   QCheckBox* solaris_check = new QCheckBox ( "Solaris", bg );
 77:   bg->resize ( 250, 100 );
 78:   bg->move( 5, 70 );
 79:   linux_check->setChecked( TRUE );
 80:
 81:   // radio buttons in a group, arranged horizontally
 82:   QButtonGroup* browserbg = new QButtonGroup ( 3, Qt::Horizontal,
 83:     "Browser of choice", this );
 84:   QRadioButton* netscape_button = new QRadioButton( "Netscape",
browserbg );
 85:   QRadioButton* konq_button = new QRadioButton( "&Konqueror",
browserbg );
 86:   QRadioButton* mozilla_button = new QRadioButton( "Mozilla",
browserbg );
 87:   browserbg->resize ( 250, 50 );
 88:   browserbg->move( 5, 180 );
 89:   konq_button->setChecked( TRUE );
 90:
 91:   // horizontal slider
 92:   QSlider* vol = new QSlider ( QSlider::Horizontal, this );
 93:   vol->resize ( 250, 30 );
 94:   vol->move ( 5, 240 );
 95:   vol->setRange ( 0, 100 );
 96:   vol->setValue ( 35 );
 97:   vol->setTickmarks ( QSlider::Below );
 98:
 99:
100:   // normal button
101:   QPushButton* normal_button = new QPushButton ( "Normal &Button",
this );
102:   normal_button->move( 5, 290 );
103:
104:   // toggle button
105:  QPushButton* toggle_button = new QPushButton ( "Toggle", this );
106:  toggle_button->setToggleButton( TRUE );
107:  toggle_button->move( 110, 290 );
```

```
108:
109:  // button with picture
110:  QPushButton* pic_button = new QPushButton ( "Floppy", this );
111:  pic_button->resize( 29, 29 );
112:  pic_button->move( 220, 290 );
113:  pic_button->setPixmap( floppy_xpm );
114:
115: }
```

Listing 4: gallery.cpp

Penting diingat bahwa widget diturunkan dari kelas `QWidget`. Kelas ini memiliki beberapa fungsi dasar penanganan widget yang umum. Dua diantaranya digunakan di `gallery.cpp`, yaitu `move` dan `resize`, masing-masing untuk memindahkan widget ke lokasi yang baru dan untuk memodifikasi ukuran (dimensi) widget tersebut. Untuk lebih rincinya, silakan merujuk ke dokumentasi Qt [3].

Sebuah *label* adalah teks statik yang diletakkan begitu saja, biasanya untuk memberikan keterangan widget lain. Pada contoh di atas misalnya, terdapat dua buah label: "Name" dan "Profession" yang masing-masing menjelaskan sedikit makna dari widget-widget di sebelahnya. Label dibentuk dengan kelas *QLabel* (dan akibatnya membutuhkan file `qlabel.h` supaya ter-include-kan). Contoh pembuatan QLabel pada listing program yang diberikan relatif sederhana, menggunakan konstruktor:

```
QLabel::QLabel ( const QString & text, QWidget * parent,
const char * name, WFlags f )
```

Di sini *text* menyatakan string yang ditampilkan sementara *parent* selalu menunjuk ke widget yang menjadi empunya label tersebut. Berikut-berikutnya akan jelas bahwa ketika mengkonstruksi sebuah widget dari sebuah window, argument *parent* hampir selalu `this`. Baik *name* dan flag *f* adalah opsional dan dapat diabaikan untuk sementara.

Perhatikan bahwa sebuah label dapat saja terdiri atas beberapa baris. Bagaimana cara membuatnya ? Cukup dengan mengatur *text* pada konstruktornya. Bila misalnya *text* ini adalah "dua baris\n teks, nih!", maka hasilnya adalah string yang disajikan dalam dua baris, semata-mata karena adanya karakter '\n' yang berfungsi sebagai *line feed*. Anda dapat mencobanya sendiri kalau penasaran.

Sebuah *edit box* adalah kotak isian yang lazimnya dimanfaatkan untuk menerima masukan dari user. Edit box ini biasanya hanya untuk string dalam satu baris saja. Dengan Qt, edit box dibuat menggunakan kelas *QLineEdit* dari file header `qlineedit.h`. Panjang maksimum string yang bisa dimasukkan adalah 32767 karakter.

Konstruksi QLineEdit malah lebih sederhana dibandingkan QLabel:

```
QLineEdit::QLineEdit ( QWidget * parent, const char * name )
```

Lagi-lagi di sini *parent* menentukan pemilik objek QLineEdit yang dibuat dan diisi `this`). Argumen *name*, karena tidak wajib, maka diacuhkan saja.

Kalaulah Anda cermat, maka bentuk konstruktor yang sebenarnya digunakan oleh program Gallery adalah:

```
QLineEdit::QLineEdit ( const QString & contents, QWidget * parent,
const char * name )
```

Tambahan argument *contents* menentukan isian awal dari edit box ini. Dengan cara ini, edit box yang baru dibuat tidak akan kosong begitu saja, seperti ditunjukkan pada contoh program yang diberikan.

Sebuah *combo box* juga berfungsi untuk menerima masukan dari user sebagaimana edit box. Bedanya adalah bahwa combo box biasanya menawarkan pilihan untuk user, biasanya dengan tombol panah di ujungnya yang jikalau diklik akan mendaftarkan item-item yang dapat dipilih. Bahkan, bisa jadi sebuah combo box cuma mengijinkan masukan dari pilihan item yang tersedia dan tidak memungkinkan input sembarang.

Bagaimana menghasilkan combo box ? Gunakan kelas *QComboBox* dari file header `qcombobox.h` dengan pilihan konstruktor:

```
QComboBox::QComboBox ( bool rw, QWidget * parent=0, const char * name )
```

atau yang lebih pendek:

```
QComboBox::QComboBox ( QWidget * parent=0, const char * name )
```

Argumen *rw* menentukan apakah combo box yang dibuat bersifat *read-and-write*. Bila diisi FALSE, maka yang terjadi adalah *read-only*, yaitu bahwa combo box hanya menerima input dari pilihan item yang disediakan saja.

Salah satu fungsi anggota paling penting dari QComboBox adalah `insertItem`. Sebagaimana dicontohkan pada `gallery.cpp` di atas, fungsi ini digunakan untuk menambahkan (atau lebih tepatnya menyisipkan) item baru pada daftar pilihan combo box tersebut.

Dua widget lain yang tidak kalah seringnya dipergunakan adalah *radio button* dan *check box*. Masing-masing sering digunakan untuk mengaktifkan atau menonaktifkan pilihan tertentu, dengan perbedaan kecil bahwa umumnya radio button berlaku untuk pilihan yang eksklusif (hanya satu di antaranya yang aktif). Masing-masing widget ini dibentuk dari kelas *QRadioButton* (file header `qradiobutton.h`) dan kelas *QCheckBox* (file header `qcheckbox.h`). Perhatikan konstruktornya di bawah ini:

```
QRadioButton::QRadioButton ( const QString & text, QWidget * parent,
const char * name )
QCheckBox::QCheckBox ( const QString & text, QWidget * parent,
const char * name )
```

Sering sekali secara visual sejumlah radio button atau check box digabungkan dalam kelompok tertentu. Dengan Qt, hal ini dapat dilakukan menggunakan kelas *QButtonGroup* dengan konstruktor:

```
QButtonGroup::QButtonGroup ( int strips, Orientation orientation,
QWidget * parent, const char * name )
```

(Jangan lupa untuk menyertakan file header `qbuttongroup.h`).

Di sini *strip* menentukan jumlah radio button atau check box yang dikelompokkan, *orientation* mengatur orientasi, bisa bernilai `Qt::horizontal` atau `Qt::vertical`, masing-masing untuk arah mendatar dan tegak.

Dengan mengumpulkan radio button atau check box dalam sebuah grup, maka argument *parent* pada konstruksi objeknya tidak bisa lagi menunjuk `this`. Hal ini jelas misalnya dari pembuatan check box dengan caption "Linux" pada contoh program yang diberikan:

```
QCheckBox* linux_check = new QCheckBox ( "Linux", bg );
```

Jelas bahwa *parent* kini adalah `bg` yang tidak lain tidak bukan adalah objek dari kelas QButtonGroup yang digunakan untuk mengumpulkan tiga buah check box, masing-masing dengan caption "Linux", "BSD", dan "Solaris". Hal yang sama berlaku juga untuk group Browser, yang kali ini membawahi tiga buat radio button yang tersusun horizontal.

Fungsi `setChecked` dari kelas QRadioButton dan QCheckBox dapat digunakan untuk mengaktifkan ataupun menonaktifkan widget yang dimaksud. Khusus untuk radio button, bila salah satunya (dalam kelompok yang sama) diaktifkan (`setChecked( TRUE )`), maka yang lainnya akan otomatis dinonaktifkan.

Widget penting lainnya adalah *slider*, yaitu sebuah penggeser yang memberikan nilai berdasarkan sebuah skala ukuran yang tetap. Slider semacam ini cocok misalnya untuk pengatur volume audio. Qt menyediakan kelas QSlider (dan file header `qslider.h`) untuk menghasilkan objek berupa sebuah slider. Simak konstruktornya:

```
QSlider::QSlider ( Orientation orientation, QWidget * parent,
const char * name )
```

Di sini *orientation* menentukan apakah slider tersebut mendatar (bila bernilai `QSlider::Horizontal`) ataukah tegak (untuk `QSlider::Vertical`).

Setelah objek slider dikonstruksi, fungsi `setRange` dapat dijalankan untuk mengatur rentang nilai yang bisa diatur dari penunjuk slider. Menempatkan penunjuk pada posisi awal bisa dilakukan dengan fungsi `setValue`.

Widget terakhir yang ditunjukkan program Gallery adalah button. Terdapat tiga variasi dari button yang disajikan, yaitu normal, toggle, serta button dengan bitmap. Button yang normal tentulah tidak asing lagi, mengingat button ini telah diperkenalkan sebelumnya (program qclick). Button dengan kemampuan `toggle` dibuat dengan cara memanggil fungsi `setToggleButton( TRUE )`. Jadi cukup sederhana dan tidak ada tambahan sesuatu yang lain. Pun button dengan pixmap tidak lain adalah button biasa yang diberikan bitmap, kali ini dengan fungsi `setPixmap`. Program Gallery mencontohkan pixmap

(berupa gambar disket) dalam format XPM, namun sesungguhnya Qt mendukung pula format-format lain seperti BMP, PNG, GIF, dan JPEG.

Sekedar informasi tambahan, XPM adalah format pixmap untuk X. Tidak umum seperti GIF, BMP, JPEG, dan PNG yang merupakan format data binary, XPM adalah dalam teks ASCII biasa (Anda dapat memodifikasinya dengan editor teks). Hal ini menyebabkan file `*.xpm` dapat begitu saja di-include-kan dalam program dan pixmapnya bisa langsung digunakan (dan karenanya menjadi pilihan tepat untuk sebuah icon ataupun pixmap kecil pada button). Untuk mengkonversi dari format lain ke XPM ataupun sebaliknya, paling gampang menggunakan GIMP [4].

Patut dicatat bahwa file program `gallery.cpp` memerlukan file `gallery.moc`. File terakhir yang akan di-include-kan dalam program ini dapat dengan menjalankan MOC (*Meta Object Compiler*). Akan dilihat bahwa file `.moc` akan dihasilkan secara otomatik pada saat kompilasi.

File `main.cpp` akan merangkaikan user interface yang dibuat sebelumnya ke dalam sebuah program utuh, sebagaimana ditunjukkan berikut ini:

```
 1: // main.cpp - part of Gallery
 2: #include <qapplication.h>:
 3: #include "gallery.h"
 4:
 5: int main( int argc, char ** argv )
 6: {
 7:   QApplication a( argc, argv );
 8:   Gallery g;
 9:   a.setMainWidget( &g );
10:   g.show();
11:   return a.exec();
12: }
```

Listing 5: main.cpp

Melakukan Kompilasi

Melakukan proses compile dan link secara manual akan amat menjemukan. Dengan bantuan `make`, semuanya bisa menjadi lebih sederhana. Makefile yang diberikan di sini relatif generik dan bisa diadaptasi dengan mudah untuk program-program Qt lainnya:

```
 1: QTDIR     = /usr/lib/qt2
 2: QTLIB     = $(QTDIR)/lib
 3: QTINCLUDE = $(QTDIR)/include
 4: QTMOC     = $(QTDIR)/bin/moc
 5:
 6: TARGET    = gallery
 7: OBJECTS   = gallery.o main.o
 8: MOCS      = gallery.moc
 9:
10: CC        = g++
11: CFLAGS    = -Wall -O2
12: LIBS      = -lqt
```

```
 13:
 14: $(TARGET): $(MOCS) $(OBJECTS)
 15:    $(CC) $(LFLAGS) -o $(TARGET) $(OBJECTS) $(LIBS) -L$(QTLIB)
-I$(QTINCLUDE)
 16:
 17: %.o: %.cpp
 18:    $(CC) $(CFLAGS) -I$(QTINCLUDE) -c $< -o $@
 19:
 20: %.moc: %.h
 21:    $(QTMOC) $< -o $@
 22:
 23: clean:
 24:    rm $(OBJECTS) $(TARGET) $(MOCS)
```

Listing 6: Makefile untuk Gallery

Bila Anda masih kurang jelas tentang penggunaan Makefile, silakan merujuk ke manual GNU Make [5] (atau dengan menggunakan perintah `info make`). Bagi yang ingin lebih canggih lagi, kelak akan digunakan autoconf dan automake sebagai pengganti Makefile

Sekarang kompilasi program Gallery dengan mengetikkan: `make`. Bila tidak ada sesuatu yang salah, maka akan segera tercipta file executable `gallery`. Jika file ini dieksekusi, akan tampil sebuah window yang kira-kira serupa sebagaimana screenshot yang telah ditunjukkan.

Bagi yang teliti, dari Makefile di atas, terlihat aturan (*rule*) yang digunakan untuk menghasilkan file `gallery.moc` yang sudah disinggung sebelumnya. Secara manual, file ini bisa dihasilkan dengan menjalankan perintah:

```
moc gallery.h -o gallery.moc
```

Utility `moc` sendiri merupakan bagian dari Qt dan sudah tersedia ketika Qt pertama kali diinstal.

Selamat ber-Qt-ria !

Tanya Jawab

Tanya: Mengapa Qt dan bukan GTK ?
Jawab: Karena Qt adalah fondasi dari KDE. Bila sebaliknya Anda ingin menguasai pemrograman di GNOME, maka gunakanlah GTK. Lepas dari hal ini, karena Qt sendiri tersedia untuk Unix/X11, Windows, dan Mac(segera), maka Qt adalah pilihan tepat untuk mengembangkan aplikasi yang *cross-platform*. Keuntungan Qt lainnya adalah API yang beragam dan mudah dipelajari, serta ketersediaan dokumentasi yang lengkap.

Tanya: Bila saya membuat program dengan Qt, apakah program ini hanya dapat dijalankan di KDE ?
Jawab: Tidak. Selama terdapat library Qt pada sistem yang digunakan, maka program tersebut tetap dapat dijalankan, tidak perduli apakah user menggunakan KDE, GNOME, WindowMaker, twm, atau window manager lainnya.

Tanya: Setahu saya KDE 2 menggunakan Qt 2.x. Kalau demikian, apakah program saya yang juga berbasis Qt 2.x bisa dijalankan di KDE 1.x ?
Jawab: Bila Anda cukup cermat, berdasarkan pertanyaan sebelumnya, maka jawabannya adalah: ya. Perhatikan bahwa screenshot contoh program `qhello` pada tulisan ini diambil di SuSE Linux 7.0, masih dengan KDE 1.1.2.

Tanya: Nah, sekarang bila program Qt saya mau digunakan oleh user yang tidak mempunyai library Qt, bagaimana caranya ?
Jawab: Gunakan *static-linking* yang akan memaksa Qt ikut serta pada executable yang dihasilkan (ini adalah salah satu yang digunakan di Opera for Linux).

Tanya: Alih-alih C++, bisakah saya menggunakan ANSI C untuk memanfaatkan Qt ?
Jawab: Tentu saja, karena sudah tersedia *binding* dari Qt untuk C "murni" (bukan C++). Jangankan C, Anda pun bisa menggunakan Perl, Python, Ruby, dan bahkan Java.

Tanya: Bagaimana perbedaan konsep *signal/slot* dengan *callback* ?
Jawab: Signal/slot mempunyai keuntungan bahwa penggunaannya lebih sederhana dibandingkan callback. Untuk pemrograman berorientasi objek (OOP), hal ini menjadi lebih menyederhanakan persoalan. Lagipula, signal/slot bersifat *typesafe* dalam arti bahwa slot hanya akan menerima signal yang mempunyai nama serta argumen yang sama persis dengan yang sebelumnya didaftarkan.

Tanya: Dapatkah lebih dari satu slot dihubungkan ke satu signal ?
Jawab: Tidak ada batasan mengenai hal ini. Satu signal dapat saja digunakan untuk memicu beberapa slot (dan kemudian menghasilkan sejumlah event). Demikian juga, satu slot boleh saja diaktifkan oleh sejumlah signal yang berbeda.

Tanya: Pada contoh program `qhello` dan `qclick`, mengapa Meta Object Compiler tidak digunakan ?
Jawab: Hal ini karena tidak ada `Q_OBJECT` pada source codenya. Mudah dipahami bahwa untuk program sesederhana `qhello`, tidak perlu dibuat sebuah widget baru, yang berarti juga tidak ada kelas baru yang memanfaatkan `Q_OBJECT`.

Tanya: Saya terus gagal untuk melakukan compile.
Jawab: Periksa source code Anda (file `gallery.cpp`). Kemungkinan Anda meng-include-kan `gallery.h`, seharusnya adalah `gallery.moc`.

Tanya: Mengapa tidak digunakan `tmake` yang sudah tersedia pada Qt untuk menghasilkan Makefile ?
Jawab: Program Gallery relatif sederhana sehingga Makefile mudah saja dibuat secara manual. Tentu bila Anda menginginkan, `tmake` dapat saja dipergunakan dengan sedikit penyesuaian.

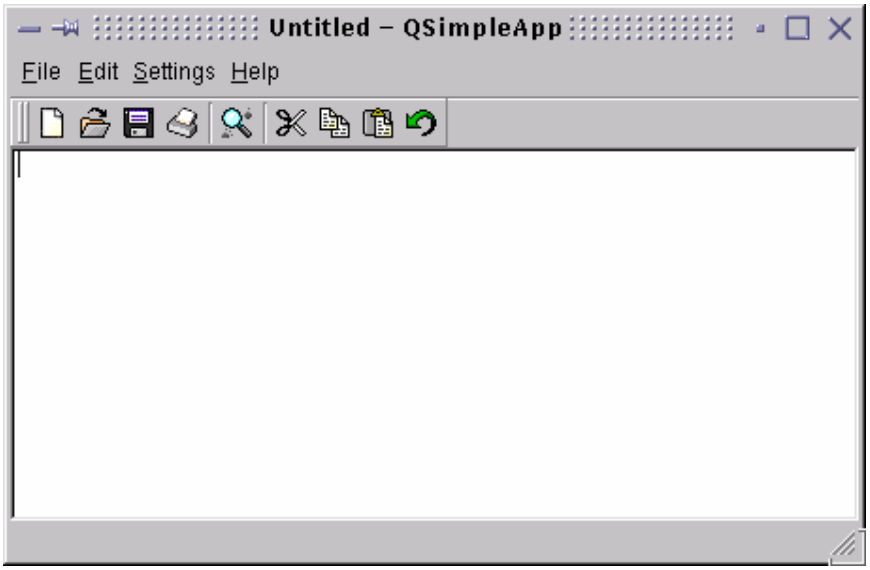
Tanya: Ada begitu banyak widget yang disediakan Qt. Bagaimanakah saya bisa mempelajari semuanya ?
Jawab: Gampangnya adalah dengan merujuk ke dokumentasi Qt. Biasanya ini bisa dijumpai di `/usr/lib/qt2/doc` (atau di tempat lain, sesuai lokasi instalasi Qt). *Tips*: untuk cepat mencapai lokasi ini, gunakan fasilitas Bookmark di Konqueror.

Tutorial Qt (bagian 2)

Setelah berkenalan dengan Qt dan melihat bagaimana cara mengkomposisikan widget-widget menjadi suatu bentuk user-inteface, akan menarik untuk menengok metoda yang harus dilakukan untuk membangun sebuah window utama program (lengkap dengan toolbar dan menu) sebagaimana lazimnya aplikasi yang berbasis grafis (GUI).

Menu, Toolbar, dan Statusbar

Sudah barang tentu program "Hello World" hanyalah sekedar program demonstrasi yang terlalu bersahaja. Berikut ini, akan diulas bagaimana membangun program aplikasi sebagaimana umumnya, yaitu mencakup pembuatan window utama, membubuhkan menu, serta menambahkan toolbar dan statusbar. Jangan khawatir karena dengan Qt, semuanya menjadi mudah dan sederhana. Untuk sekedar membangkitkan motivasi, di bawah ini disajikan dulu screenshot program yang akan dibahas:



Gambar 1: Screenshot Aplikasi Sederhana dengan Qt

Program `qsimpleapp` akan terdiri dari empat file source code, yaitu `qsimpleapp.h`, `qsimpleapp.cpp`, `main.cpp`, dan `icons.h`. Sebuah `Makefile` juga dibutuhkan untuk memudahkan proses kompilasi. Berikut adalah source code untuk file-file tersebut.

```
 1: // qsimpleapp.h
 2: #ifndef __QSIMPLEAPP_H
 3: #define __QSIMPLEAPP_H
 4:
 5: class QMultiLineEdit;
 6: class QPopupMenu;
 7: class QString;
 8: class QToolBar;
 9:
10: #include <qmainwindow.h>
```

```
11:
12: class QSimpleWindow: public QMainWindow
13: {
14:     Q_OBJECT
15:
16: public:
17:     QSimpleWindow();
18:
19: protected:
20:     void initMenu();
21:     void initToolbar();
22:     void load( const char *fileName );
23:
24: private slots:
25:     void fileNew();
26:     void fileOpen();
27:     void fileSave();
28:     void fileSaveAs();
29:     void filePrint();
30:     void fileClose();
31:
32:     void editUndo();
33:     void editRedo();
34:     void editCut();
35:     void editCopy();
36:     void editPaste();
37:     void editFind();
38:
39:     void settingsToggleToolbar();
40:     void settingsToggleStatusbar();
41:     void configure();
42:
43:     void about();
44:     void aboutQt();
45:
46: private:
47:
48:     QString filename;
49:     QMultiLineEdit *e;
50:     QToolBar *toolbar;
51:     int menuid_toolbar, menuid_statusbar;
52:
53: };
54:
55: #endif
```

Listing 1: qsimpleapp.h

File `qsimpleapp.h` adalah file header utama. Pada file ini didefinisikan kelas `QSimpleWindow` yang diturunkan dari `QMainWindow`. Nantinya instance dari objek ini kelas ini akan dijadikan widget utama program.

Sekedar penjelasan ringkas, fungsi `initMenu()` adalah untuk menginisialisasi menu, fungsi `initToolbar()` untuk mengatur toolbar, dan fungsi `load()` dipergunakan untuk membuka dan mengambil dokumen dari file tertentu.

Sejumlah slot baru juga dideklarasikan di sini, bisa dilihat setelah `private slots:`. Masing-masing slot ini akan dikoneksikan agar dapat menerima signal yang berasal dari menu. Dengan demikian, slot `fileNew()` akan dipanggil bila ada signal yang dihasilkan dengan memilih New dari menu File, slot `slotOpen` untuk Open dari menu File, dan seterusnya.

Implementasi dari kelas `QSimpleWindow` sendiri terletak di file `qsimpleapp.cpp` berikut ini:

```
 1: // qsimpleapp.cpp - Qt example program
 2: #include <qpixmap.h>
 3: #include <qtoolbar.h>
 4: #include <qtoolbutton.h>
 5: #include <qpopupmenu.h>
 6: #include <qmenubar.h>
 7: #include <qkeycode.h>
 8: #include <qmultilineedit.h>
 9: #include <qfileinfo.h>
10: #include <qstatusbar.h>
11: #include <qmessagebox.h>
12: #include <qapplication.h>
13: #include <qaccel.h>
14:
15: #include "qsimpleapp.moc"
16:
17: #include "icons.h"
18:
19: QSimpleWindow::QSimpleWindow()
20:     : QMainWindow( 0, "QSimpleWindow", WDestructiveClose )
21: {
22:   initMenu();
23:   initToolbar();
24:
25:   e = new QMultiLineEdit( this, "editor" );
26:   e->setFocus();
27:   setCentralWidget( e );
28:   statusBar()->message( "Ready", 2000 );
29:   resize( 450, 600 );
30:
31:   menuBar()->setItemChecked( menuid_toolbar, TRUE );
32:   menuBar()->setItemChecked( menuid_statusbar, TRUE );
33:
34:   setCaption("Untitled - QSimpleApp");
35: }
36:
37:
38: // initialize menu
39: void QSimpleWindow::initMenu()
40: {
41:     QPopupMenu * file = new QPopupMenu( this );
42:     menuBar()->insertItem( "&File", file );
43:
44:     file->insertItem( QPixmap(filenew_xpm), "&New", this,
SLOT(fileNew()), CTRL+Key_N );
45:     file->insertItem( QPixmap(fileopen_xpm), "&Open",  this,
SLOT(fileOpen()), CTRL+Key_O );
```

```
 46:     file->insertItem( QPixmap(filesave_xpm), "&Save",  this,
SLOT(fileSave()), CTRL+Key_S );
 47:     file->insertItem( "Save &as...", this, SLOT(fileSaveAs()) );
 48:     file->insertSeparator();
 49:     file->insertItem( QPixmap(fileprint_xpm), "&Print...", this,
SLOT(filePrint()), CTRL+Key_P );
 50:     file->insertSeparator();
 51:     file->insertItem( QPixmap(fileclose_xpm), "&Close", this,
SLOT(fileClose()), CTRL+Key_W );
 52:     file->insertItem( "&Quit", qApp, SLOT(closeAllWindows() ),
CTRL+Key_Q );
 53:
 54:     QPopupMenu * edit = new QPopupMenu ( this );
 55:     menuBar()->insertSeparator();
 56:     menuBar()->insertItem("&Edit", edit);
 57:     edit->insertItem( QPixmap(editundo_xpm), "Und&o", this,
SLOT(editUndo()), CTRL+Key_U );
 58:     edit->insertItem( "Redo", this, SLOT(editRedo()) );
 59:     edit->insertSeparator();
 60:     edit->insertItem( QPixmap(editcut_xpm), "C&ut", this,
SLOT(editCut()), CTRL+Key_X);
 61:     edit->insertItem( QPixmap(editcopy_xpm), "&Copy", this,
SLOT(editCopy()), CTRL+Key_C );
 62:     edit->insertItem( QPixmap(editpaste_xpm), "&Paste", this,
SLOT(editPaste()), CTRL+Key_V );
 63:     edit->insertSeparator();
 64:     edit->insertItem( QPixmap(editfind_xpm), "&Find...", this,
SLOT(editFind()) );
 65:
 66:     QPopupMenu *settings = new QPopupMenu ( this );
 67:     menuBar()->insertSeparator();
 68:     menuBar()->insertItem("&Settings", settings);
 69:     menuid_toolbar = settings->insertItem ( "Show &Toolbar", this,
SLOT(settingsToggleToolbar()) );
 70:     menuid_statusbar = settings->insertItem ( "Show &Statusbar",
this, SLOT(settingsToggleStatusbar()) );
 71:     settings->insertSeparator();
 72:     settings->insertItem ( QPixmap(configure_xpm), "Configure...",
this, SLOT(configure()) );
 73:
 74:     QPopupMenu * help = new QPopupMenu( this );
 75:     menuBar()->insertSeparator();
 76:     menuBar()->insertItem( "&Help", help );
 77:     help->insertItem( QPixmap(helpabout_xpm), "&About...", this,
SLOT(about())  );
 78:     help->insertItem( "About &Qt...", this, SLOT(aboutQt()) );
 79: }
 80:
 81: // initialize toolbar
 82: void QSimpleWindow::initToolbar()
 83: {
 84:     toolbar = new QToolBar( this );
 85:
 86:     new QToolButton( QPixmap(filenew_xpm), "New", QString::null,
 87:                 this, SLOT(fileNew()), toolbar, "open file" );
 88:     new QToolButton( QPixmap(fileopen_xpm), "Open", QString::null,
 89:                 this, SLOT(fileOpen()), toolbar, "open file" );
 90:     new QToolButton( QPixmap(filesave_xpm), "Save", QString::null,
 91:                 this, SLOT(fileSave()), toolbar, "save file" );
```

```
 92:
 93:     new QToolButton( QPixmap(fileprint_xpm), "Print",
QString::null,
 94:                 this, SLOT(filePrint()), toolbar, "print file" );
 95:
 96:     toolbar->addSeparator();
 97:
 98:     new QToolButton( QPixmap(editfind_xpm), "Find", QString::null,
 99:                 this, SLOT(editFind()), toolbar, "find");
100:
101:     toolbar->addSeparator();
102:
103:     new QToolButton( QPixmap(editcut_xpm), "Cut", QString::null,
104:                 this, SLOT(editCut()), toolbar, "cut");
105:     new QToolButton( QPixmap(editcopy_xpm), "Copy", QString::null,
106:                 this, SLOT(editCopy()), toolbar, "copy");
107:     new QToolButton( QPixmap(editpaste_xpm), "Paste",
QString::null,
108:                 this, SLOT(editPaste()), toolbar, "paste");
109:     new QToolButton( QPixmap(editundo_xpm), "Undo", QString::null,
110:                 this, SLOT(editUndo()), toolbar, "undo");
111:
112: }
113:
114: // creates a new blank file
115: void QSimpleWindow::fileNew()
116: {
117:     QSimpleWindow *f = new QSimpleWindow;
118:     f->show();
119: }
120:
121: // opens and loads from file
122: void QSimpleWindow::fileOpen()
123: {
124: }
125:
126: // loads given file
127: void QSimpleWindow::load( const char *fileName )
128: {
129:   // do file loading here
130:
131:   // update caption and titlebar
132:   QFileInfo fi(fileName);
133:   QString fn = fi.fileName();
134:   setCaption( fn  + " - QSimpleApp" );
135:   statusBar()->message( fn + " loaded", 2000 );
136: }
137:
138: // saves active
139: void QSimpleWindow::fileSave()
140: {
141:   if ( filename.isEmpty() ) {
142:     fileSaveAs();
143:     return;
144:   }
145:
146:
147:   // do file saving here
148:
```

```
149:
150:   // update caption and statusbar
151:   QFileInfo fi(filename);
152:   QString fn = fi.fileName();
153:   setCaption( fn + " - QSimpleApp" );
154:   statusBar()->message( fn + " saved", 2000 );
155: }
156:
157: // saves to a different file
158: void QSimpleWindow::fileSaveAs()
159: {
160: }
161:
162: // prints current document
163: void QSimpleWindow::filePrint()
164: {
165: }
166:
167: // closes this window
168: void QSimpleWindow::fileClose()
169: {
170:   close();
171: }
172:
173: // reverts last action
174: void QSimpleWindow::editUndo()
175: {
176: }
177:
178: // redoes last action
179: void QSimpleWindow::editRedo()
180: {
181: }
182:
183:
184: void QSimpleWindow::editCut()
185: {
186: }
187:
188: void QSimpleWindow::editCopy()
189: {
190: }
191:
192: void QSimpleWindow::editPaste()
193: {
194: }
195:
196: void QSimpleWindow::editFind()
197: {
198: }
199:
200: void QSimpleWindow::settingsToggleToolbar()
201: {
202:   bool isToolbar = menuBar()->isItemChecked( menuid_toolbar );
203:   isToolbar = !isToolbar;
204:   if(isToolbar) toolbar->show(); else toolbar->hide();
205:   menuBar()->setItemChecked( menuid_toolbar, isToolbar );
206: }
207:
```

```
208: void QSimpleWindow::settingsToggleStatusbar()
209: {
210:   bool isStatusbar = menuBar()->isItemChecked( menuid_statusbar );
211:   isStatusbar = !isStatusbar;
212:   if(isStatusbar) statusBar()->show(); else statusBar()->hide();
213:   menuBar()->setItemChecked( menuid_statusbar, isStatusbar );
214: }
215:
216: void QSimpleWindow::configure()
217: {
218: }
219:
220: void QSimpleWindow::about()
221: {
222:   QMessageBox::about( this, "QSimpleApp",
223:                 "This is a very simple framework example "
224:                 "for Qt-based application.");
225: }
226:
227: void QSimpleWindow::aboutQt()
228: {
229:     QMessageBox::aboutQt( this, "QSimpleApp" );
230: }
```

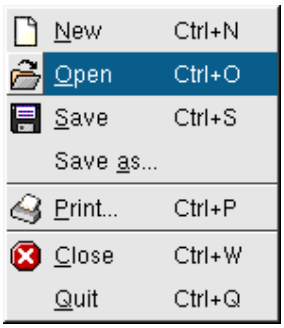
Listing 2: qsimpleapp.cpp

Pada source code di atas, bisa dilihat beberapa hal yang cukup penting.

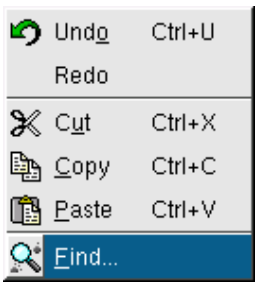
Toolbar dibuat di dalam fungsi `initToolbar()`. Dapat dilihat bahwa caranya tidak terlalu rumit, yakni cukup dengan membuat objek baru dari QToolBar, kemudian menyisipkan button yang diinginkan (dari QToolButton). Kelas QToolButton dapat dikonstruksi langsung sambil memberikan argumen yang dibutuhkan. Pada contoh ini, terdapat 6 argumen, masing-masing adalah icon yang ditampilkan, teks untuk caption, teks untuk grup, slot yang dipanggil jika button diklik (misalnya `SLOT(fileNew())`), toolbar yang menjadi induk button ini, serta nama buttonnya. Untuk icon, semuanya disimpan di file header `icons.h` dalam format X Pixmap. Caption untuk button biasanya tidak diperlukan kecuali toolbarnya diatur untuk menampikan icon dengan teks. Garis vertikal pembatas icon dihasilkan dengan fungsi `QToolBar::addSeparator()`

Agak sedikit berbeda, susunan menu justru dibangun dari beberapa pop-up menu menggunakan kelas QPopupMenu yang disisipkan ke menu utama. Kelas QMainWindow menyediakan fungsi `menuBar()` untuk mendapatkan menu utama ini. Dengan sendirinya, pop-up menu yang dihasilkan tinggal disisipkan saja, menggunakan `QMenuBar::insertItem`. Sementara itu, menyusun pop-up menu sendiri tidaklah sulit, sebagaimana ditunjukkan dalam fungsi `initMenu`. Patut dicatat bahwa untuk item menu yang tidak memiliki icon khusus, argumen pertama untuk gambar icon bisa diabaikan saja. Pada saat membuat item-item menu, sudah langsung ditetapkan slot yang akan menerima signal dari item menu tersebut berikut kombinasi tombol shortcutnya, sebagai contoh `SLOT(fileSave())` untuk menu File -> Save dengan shortcut Ctrl+S. Biasanya hal ini juga disesuaikan dengan caption dari masing-masing item yakni dengan menggarisbawahi karakter shortcutnya. Dalam contoh sebelumnya, hal ini berarti

karakter S pada item Save harus ditampilkan secara *underline* yang bisa dilakukan dengan melekatkan tanda & di depannya (ditulis sebagai "&Save").

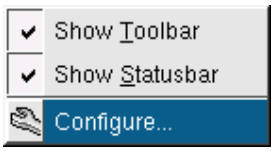


Gambar 2: Menu File



Gambar 3: Menu Edit

Menu Settings memerlukan penanganan khusus. Diinginkan agar user dapat menampilkan atau menyembunyikan toolbar dan statusbar menggunakan item "Show toolbar" dan "Shot statusbar". Hal ini bisa dilakukan dengan mengoneksikan signal dari item menu yang dimaksud dengan slot `settingsToggleToolbar()` dan `settingsToggleStatusbar()`. Perlu ditambilkan juga semacam *checkmark* pada item menunya sendiri, karenanya id untuk kedua item ini disimpan pada variabel `menuid_toolbar` dan `menuid_statusbar` supaya selanjutnya bisa diatur dengan fungsi `QMenuBar::isItemChecked` dan `QMenuBar::setItemChecked`



Gambar 4: Menu Settings

Bagaimana dengan status bar ? Beruntunglah bahwa `QMainWindow` juga sudah menyediakan fungsi `statusBar()` untuk mendapatkan status bar window yang aktif. Dengan demikian, pesan-pesan tertentu dengan gampang ditampilkan menggunakan `QStatusBar::message()`. Untuk panggilan fungsi ini, parameter pertama adalah teksnya dan parameter kedua (yang sebenarnya opsional) adalah durasi teks tersebut ditampilkan (dalam mikrodetik). Dengan cara ini, mudah menyajikan pesan tertentu pada

status bar yang akan menghilang sendiri setelah beberapa saat. Bila diinginkan teks di status bar ini tidak menghilangkan, cukup gunakan satu parameter saja ketika memanggil `QStatusBar::message()`.

QMessageBox adalah kelas statik yang bisa dipergunakan untuk menampilkan pesan tertentu. Pada contoh ini, QMessageBox digunakan di slot `about()` dan `aboutQt()`.

Menarik juga untuk disimak betapa sederhananya instruksi untuk menghasilkan window baru, yaitu pada slot `fileNew`. Cukup buat satu instance baru dari window tersebut !

Slot-slot lainnya, yaitu `fileOpen`, `fileSave`, dan seterusnya masih tidak berisi apa-apa karena memang belum ada fungsionalitas khusus dari contoh program ini. Anda tentu bisa melengkapinya sendiri jika diinginkan.

Bila Anda cukup jeli, terlihat bahwa alih-alih `qsimpleapp.h`, yang di-include-kan di `qsimpleapp.cpp` adalah `qsimpleapp.moc`. Hal ini karena `qsimpleapp.cpp` membutuhkan kelas QSimpleApp yang sudah dimodifikasi dengan Meta Object Compiler (moc). File `qsimpleapp.moc` sendiri dihasilkan oleh Meta Object Compiler (`moc`) dari file `qsimpleapp.h`.

Kelas QSimpleWindow yang demikian panjang lebar tidak akan berarti apa-apa bila tidak dipergunakan. Di bawah ini adalah `main.cpp` yang membuat aplikasi sesungguhnya dengan `QSimpleWindow` sebagai widget utamanya.

```
 1: // Qt Simple Application
 2: #include <qapplication.h>
 3: #include "qsimpleapp.h"
 4:
 5: int main( int argc, char ** argv )
 6: {
 7:   QApplication a( argc, argv );
 8:
 9:   QSimpleWindow * mw = new QSimpleWindow();
10:   mw->show();
11:
12:   a.connect( &a, SIGNAL(lastWindowClosed()), &a, SLOT(quit()) );
13:
14:   return a.exec();
15: }
```

Listing 3: main.cpp

Tidak kalah penting adalah `icons.h`. Icon-icon yang (mudah-mudahan) indah dari screenshot program di atas, baik untuk menu maupun toolbar, berasal dari file header ini (yang di-include-kan dari `qsimpleapp.cpp`).

```
1: // icons.h - part of Qt example program
2:
3: #ifndef __ICONS_H
4: #define __ICONS_H
5:
```

```
 6: const static char * configure_xpm[] = {
 7: "16 16 10 1",
 8: "  c None",
 9: ". c #FFFFFF",
10: "+ c #000000",
11: "@ c #DCDCDC",
12: "# c #585858",
13: "$ c #A0A0A0",
14: "% c #FFFFFF",
15: "& c #C3C3C3",
16: "* c #303030",
17: "= c #808080",
18: "   ++++++       ",
19: "   +=#@@$++     ",
20: "    +&##@&$+    ",
21: "+++  ++&#@$+    ",
22: "+%$++  +#@@$+   ",
23: "+%@@@+##@@@$+   ",
24: "+#%%@@@@@@@@$++ ",
25: "*%##%%@@@@@@@$$+",
26: "+&@%####$%%@@@@$",
27: " ++&@&&$###%%@@@",
28: "   +++$&&&$##%%@",
29: "      ++++@@$##%",
30: "          ++@@$#",
31: "            ++@@",
32: "              ++",
33: "                "};
34:
35: const static char * editcopy_xpm[] = {
36: "16 16 8 1",
37: "  c None",
38: ". c #FFFFFF",
39: "+ c #000000",
40: "@ c #FFFFFF",
41: "# c #303030",
42: "$ c #FFFFC0",
43: "% c #A0A0A0",
44: "& c #DCDCDC",
45: "++++++          ",
46: "+@@@@++         ",
47: "+@@@@+$+        ",
48: "+@##@+@$+       ",
49: "+@@@@+++++++    ",
50: "+@##@@%%%+@++   ",
51: "+@@@@@$@$+@+&+  ",
52: "+@###@##$+@+@&+ ",
53: "+@@@$@$$$+@+++++",
54: "+@##@###$+@@%%%+",
55: "+@$@$$$$$+@@@@&+",
56: "++++++++++##@##@+",
57: "      +@@@@@@@@+",
58: "      +@##@###@+",
59: "      +@@@@@@@@+",
60: "      ++++++++++"};
61:
62: const static char * editcut_xpm[] = {
63: "16 16 6 1",
64: "  c None",
```

```
 65: ". c #FFFFFF",
 66: "+ c #000000",
 67: "@ c #808080",
 68: "# c #DCDCDC",
 69: "$ c #FFFFFF",
 70: "                ",
 71: " ++++         ++ ",
 72: "++  ++       +$#+",
 73: "++   ++     +$#+ ",
 74: " ++   +    +$#+  ",
 75: "  ++++++ +$#+    ",
 76: "    +++++$#+     ",
 77: "      ++$#+      ",
 78: "      +$#++      ",
 79: "    +++++@#+     ",
 80: "  ++++++ +$#+    ",
 81: " ++   +    +$#+  ",
 82: "++   ++     +$#+ ",
 83: "++  ++       +$#+",
 84: " ++++         ++ ",
 85: "                "};
 86:
 87: const static char * editfind_xpm[] = {
 88: "16 16 13 1",
 89: "  c None",
 90: ". c #FFFFFF",
 91: "+ c #000000",
 92: "@ c #C0FFFF",
 93: "# c #585858",
 94: "$ c #FFFFFF",
 95: "% c #00C0C0",
 96: "& c #303030",
 97: "* c #808080",
 98: "= c #C3C3C3",
 99: "- c #008080",
100: "; c #00FFFF",
101: "gt;       c #004040",
102: "    +++       # ",
103: "  ++%@%++       ",
104: " +-@@@@@-+   ** ",
105: " +@@$$$@@+  *###",
106: "+%@$$$@@@%+ ###&",
107: "+@@$$@@@@@+  && ",
108: "+%@$@@@@@%+     ",
109: " +@@@@@@@+      ",
110: " +-@@@@@-+;     ",
111: "  ++%@%++%++    ",
112: "    +++ gt;+#=+   ",
113: "  #      +&#=+  ",
114: "    #     +&#=+ ",
115: " **        +&#=+",
116: "*### #      +&#+",
117: "###&         ++ "};
118:
119: const static char * editpaste_xpm[] = {
120: "16 16 9 1",
121: "  c None",
122: ". c #FFFFFF",
123: "+ c #000000",
```

```
124: "@ c #A0A0A0",
125: "# c #FFFFFF",
126: "$ c #FFFFC0",
127: "% c #303030",
128: "& c #C0C000",
129: "* c #DCDCDC",
130: "       ++       ",
131: " ++++++&&+++++  ",
132: "+@@@@+&++&+@@@+ ",
133: "+*@@+&&&&&&+@@+ ",
134: "+*@@++++++++@@+ ",
135: "+*@@@@+######++@+ ",
136: "+*@@@@+####+$++ ",
137: "+*@@@@+#%%#+#$++",
138: "+*@@@@+##########+",
139: "+*@@@@+#%%##@@@+",
140: "+*@@@@+#####$#$+",
141: "+*@@@@+#%%%#%%$+",
142: "+*@@@@+###$#$$$+",
143: "+@@@@@+#%%#%%%$+",
144: " ++++++#$#$$$$$+",
145: "      ++++++++++"};
146:
147: const static char * editundo_xpm[] = {
148: "16 16 6 1",
149: "  c None",
150: ". c #FFFFFF",
151: "+ c #000000",
152: "@ c #00C000",
153: "# c #008000",
154: "$ c #004000",
155: "                ",
156: "       ++++     ",
157: "     ++@@@@++   ",
158: "+   +@@@@####+  ",
159: "+# +@@@##++$##+ ",
160: "+@+@@@#++  ++$+ ",
161: "+@@@@#+      +$+",
162: "+@@@@+       +++",
163: "+@@@@+       +++",
164: "+######      +++",
165: "++++++++     ++ ",
166: "            +++ ",
167: "           +++  ",
168: "         ++++   ",
169: "        +++     ",
170: "                "};
171:
172: const static char * fileclose_xpm[] = {
173: "16 16 7 1",
174: "  c None",
175: ". c #800000",
176: "+ c #C00000",
177: "@ c #000000",
178: "# c #FFFFFF",
179: "$ c #800000",
180: "% c #FFC0C0",
181: "    @@@@@@@@    ",
182: "   @%%%%%%%%@   ",
```

```
183: "  @%%+++++++$@  ",
184: " @%%+++++++++$@ ",
185: "@%%+#######++$@",
186: "@%++#####++#++$@",
187: "@%+++########+++$@",
188: "@%++++#########++++$@",
189: "@%++++#########++++$@",
190: "@%+++#########+++$@",
191: "@%++#####++#++$@",
192: "@%++########++#++$@",
193: " @$++++++++++$@ ",
194: "  @$++++++++$@  ",
195: "   @$$$$$$$$$@   ",
196: "    @@@@@@@@    "};
197:
198: const static char * filenew_xpm[] = {
199: "16 16 11 1",
200: "  c None",
201: ". c #5D5D5D",
202: "+ c #FFFFFF",
203: "@ c #000000",
204: "# c #FFFFC0",
205: "$ c #DCDCDC",
206: "% c #303030",
207: "& c #585858",
208: "* c #C3C3C3",
209: "= c #A0A0A0",
210: "- c #400000",
211: "  @@%@@@%@@     ",
212: "  %$$$$$*%*@    ",
213: "  @$+++++&+$@   ",
214: "  @$+++++&$+*@  ",
215: "  @$+++++&&&%@  ",
216: "  @$++++++$*=@  ",
217: "  @$++++++#+$@  ",
218: "  @$+++++++++@  ",
219: "  @$++++#+#+#@  ",
220: "  @$+++++++#+@  ",
221: "  @$++#+#+#+#@  ",
222: "  @$+++++#+#+@  ",
223: "  %$#+#+#+#+#@  ",
224: "  @$+++#+#+###@  ",
225: "  @$#+#+#+###@  ",
226: "  @@@@@@-@@%@@  "};
227:
228: const static char * fileopen_xpm[] = {
229: "16 16 8 1",
230: "  c None",
231: ". c #5D5D5D",
232: "+ c #000000",
233: "@ c #A0A0A0",
234: "# c #C3C3C3",
235: "$ c #DCDCDC",
236: "% c #FFA858",
237: "& c #FFDCA8",
238: "      ++++      ",
239: "     ++ ++++ +  ",
240: "    +     ++++  ",
241: "           +++  ",
```

```
242: "          ++++  ",
243: " ++++           ",
244: "+%&&%+++++++    ",
245: "+&&&&%%%%%%+   ",
246: "+&&++++++++++++",
247: "+&%+$$$$$$$$$$+",
248: "+&+$####@#@#@@+ ",
249: "+%+$###@#@#@@@+ ",
250: "++$###@#@#@@@+  ",
251: "++$##@#@#@@@@+  ",
252: "+$##@#@#@@@@+   ",
253: "++++++++++++   "};
254:
255: const static char * fileprint_xpm[] = {
256: "16 16 12 1",
257: "  c None",
258: ". c #FFFFFF",
259: "+ c #FFFFFF",
260: "@ c #A0A0A0",
261: "# c #DCDCDC",
262: "$ c #000000",
263: "% c #585858",
264: "& c #FFDCA8",
265: "* c #C3C3C3",
266: "= c #303030",
267: "- c #808080",
268: "; c #404000",
269: "        %$%     ",
270: "       %++*%$   ",
271: "      ;++++++@% ",
272: "      %+++++++@$",
273: "    $%++++++++=$",
274: "   $-*++++++++@-$",
275: "  $@%+&+&++++%$ ",
276: " $@@@%%&+&+&@$% ",
277: "$@@@@@@%%%&+=%$ ",
278: "$##@@@@@@@%%%@@$",
279: "$######@@@@@@@%$",
280: "$##########@@@%%$",
281: " $$%########%%%$",
282: "    $$%#####%%$ ",
283: "       $$%##%$  ",
284: "          $$$   "};
285:
286: const static char * filesave_xpm[] = {
287: "16 16 11 1",
288: "  c None",
289: ". c #5D5D5D",
290: "+ c #000000",
291: "@ c #303030",
292: "# c #FFFFFF",
293: "$ c #800080",
294: "% c #DCDCDC",
295: "& c #C3C3C3",
296: "* c #000080",
297: "= c #0000FF",
298: "- c #585858",
299: "++++++++++++++++",
300: "+@+$=$*$=$*$=+@+",
```

```
301: "+#+############+++",
302: "+@+###########+@+",
303: "+@+#$*$=$*$=#+@+",
304: "+@+###########+@+",
305: "+@+#$*$=$*$*#+@+",
306: "+@+###########+@+",
307: "+@@++++++++++@@+",
308: "+@@@@@@@@@@@@@@@@+",
309: "+@@@++++++++@@@+",
310: "+@@+#%#%%%&%+@@+",
311: "+@@+%@-&%&%&+@@+",
312: "+@@+#@-%&%&&+@@+",
313: "++@+%&%&%&&&+@@+",
314: " ++++++++++++++++"};
315:
316: const static char * helpabout_xpm[] = {
317: "16 16 5 1",
318: "  c None",
319: ". c #FFFFFF",
320: "+ c #000000",
321: "@ c #0000FF",
322: "# c #000080",
323: "    +++++++     ",
324: "   +@#+++@#++   ",
325: "  +@#+   +@@#+  ",
326: "  +@+     +@#+  ",
327: "  +@#+    +@#+  ",
328: "   +@#+   +@#+  ",
329: "    ++   +@#+   ",
330: "         +@#+    ",
331: "        +@#+     ",
332: "       +@#+      ",
333: "       +@#+      ",
334: "        ++       ",
335: "                 ",
336: "        ++       ",
337: "       +@#+      ",
338: "        ++       "};
339:
340: #endif
```

Listing 4: icons.h

Proses kompilasi dilakukan dengan menggunakan Makefile. Ini adalah serupa dengan Makefile yang pernah disajikan di bagian pertama seri tulisan ini, hanya dilakukan modifikasi di bagian TARGET, OBJECTS, dan MOCS. Sekedar menyegarkan ingatan, sebagaimana telah diungkapkan di bagian pertama seri tulisan ini, Anda mungkin perlu menyesuaikan QTDIR sesuai dengan direktori instalasi Qt pada sistem Anda.

```
1: QTDIR     = /usr/lib/qt2
2: QTLIB     = $(QTDIR)/lib
3: QTINCLUDE = $(QTDIR)/include
4: QTMOC     = $(QTDIR)/bin/moc
5:
6: TARGET    = qsimpleapp
7: OBJECTS   = qsimpleapp.o main.o
8: MOCS      = qsimpleapp.moc
```

```
  9:
 10: CC        = g++
 11: CFLAGS    = -Wall -O2
 12: LIBS      = -lqt
 13:
 14: $(TARGET): $(MOCS) $(OBJECTS)
 15:    $(CC) $(LFLAGS) -o $(TARGET) $(OBJECTS) $(LIBS) -L$(QTLIB)
-I$(QTINCLUDE)
 16:
 17: %.o: %.cpp
 18:    $(CC) $(CFLAGS) -I$(QTINCLUDE) -c $< -o $@
 19:
 20: %.moc: %.h
 21:    $(QTMOC) $< -o $@
 22:
 23: clean:
 24:    rm $(OBJECTS) $(TARGET) $(MOCS)
```

Listing 5: Makefile untuk QSimpleApp

Sekarang kompilasi program QSimpleApp dengan mengetikkan: `make`. Dengan menjalankan file executable `qsimpleapp`, akan muncul sebuah window yang kira-kira serupa sebagaimana screenshot pada awal tulisan. Program tersebut tidak mempunyai fungsionalitas yang jelas alias sekedar demo. Kendati demikian, diharapkan kerangka program yang dibangun di sini dapat dipergunakan untuk mengembangkan program lain yang lebih fungsional.

Sebagai latihan, berangkat dari QSimpleApp, Anda dapat membuat editor teks dengan mudah. Untuk ini, dokumentasi Qt tentang QMultiLineEdit akan sangat menolong. Pada instalasi Qt, disertakan juga contoh program editor teks yang dapat dipelajari.

Proses Kompilasi

Dengan Makefile, kerumitan melakukan *compile-and-link* (dan juga menjalankan `moc`) tidak dijumpai karena hal ini secara otomatis dikerjakan oleh `make`. Sekedar agar dapat mengetahui apa yang sebenarnya terjadi, untuk melakukan compile program di atas secara manual (tidak menggunakan Makefile), maka berikut adalah langkah-langkahnya:

- Guna memudahkan, set dahulu variabel lingkungan yang sesuai, dalam hal ini adalah `QTDIR`, `QTINCLUDE`, dan `QTLIB`. Bisa dilakukan dengan perintah seperti ini:
- `export QTDIR=/usr/lib/qt2`
- `export QTINCLUDE=$(QTDIR)/include`
- `export QTLIB=$(QTDIR)/lib`

  Tentu saja ini harus disesuaikan lokasi instalasi Qt pada sistem Anda. Untuk shell selain `bash`, Anda juga harus mengubah untuk mengatur environment variable (lihat manual dari shell yang Anda gunakan).

- File `qsimpleapp.moc` dapat dihasilkan dengan menjalankan Meta Object Compiler seperti berikut ini:

- `$(QTDIR)/bin/moc qsimpleapp.h -o qsimpleapp.moc`
- Selanjutnya, compile kedua file cpp yang ada, sebagaimana diilustrasikan di bawah ini:
- `g++ -c qsimpleapp.cpp`
- `g++ -c main.cpp`
- Akhirnya, link semua file objek untuk menghasilkan programnya:
- `g++ -o qsimplepp qsimpleapp.o main.o -I$(QTINCLUDE) -L$(QTLIB) -lqt`

Tanya Jawab

Tanya: Saya terus gagal untuk melakukan compile.
Jawab: Periksa source code Anda (file `qsimpleapp.cpp`). Kemungkinan Anda meng-include-kan `qsimpleapp.h`, seharusnya adalah `qsimpleapp.moc`.

Tanya: Terkadang, setelah mengakhiri program, muncul pesan kesalahan "Segmentation Fault". Mengapa ?
Jawab: Masalah dependensi terkadang menyebabkan file hasil Meta Object Compiler tidak sinkron dengan program utama. Solusinya adalah melakukan kompilasi ulang semua file (gampangnya dengan menjalankan `make clean`, baru kemudian `make` seperti biasa).

Tanya: Bagaimana membuat item pada menu utama yang punya sub-menu lagi ?
Jawab:. Anda bisa menyisipkan item tersebut dengan fungsi `QPopupMenu::insertItem ( const QString & text, QPopupMenu * popup)`. Di sini `text` adalah teks untuk item tersebut sedangkan `popup` merupakan sub-menu dari item ini. Jangan lupa untuk membuat sub-menunya terlebih dahulu sebelum memanggil fungsi di atas.

Tanya: Pada GTK, tiap menu mempunyai *handle* khusus yang jika diklik akan menyebabkan menu tersebut seakan terlepas dari program utama dan mengambang (*floating menu*). Bagaimana cara menghasilkan yang semacam ini di Qt ?
Jawab: Gunakan fungsi `QPopupMenu::insertTearOffHandle()` sebelum menyisipkan item pertama pada sebuah menu (dengan demikian, *handle* ini akan diletakkan paling atas). Anda akan mendapatkan hasil persis seperti yang Anda inginkan.

Tanya: Bagaimana menyisipkan combobox di toolbar ?
Jawab: Gunakan `QMenuData::insertItem ( QWidget * widget )`. Sebagai `widget`, masukkan combobox yang dimaksud. Karena fungsi ini bisa digunakan untuk menyisipkan widget apa saja, ini berarti tidak hanya terbatas untuk combobox semata.

Tanya: Kalau kita lihat browser seperti Netscape, toolbarnya mempunyai button yang relatif lebih besar dan dilengkapi dengan teks. Apakah hal ini bisa diwujudkan di Qt ?
Jawab: Tentu saja. Untuk menghasilkan button berukuran besar, gunakan gambar (pixmap) yang ukuran lebih besar (Qt akan menyesuaikan ukuran toolbar secara otomatis). Contoh program di atas memakai pixmap kecil, hanya 16x16 piksel. Bilamana pixmapnya misalnya 24x24 piksel, Anda akan mendapatkan toolbar yang serupa dengan dengan Netscape atau program-program lain. Tentang label pada toolbar, ini bisa

diaktifkan dari fungsi `QMainWindow::setUsesTextLabel( bool enable )`. Panggil dari konstruktor window dengan `enable` diset ke `TRUE`. Mudah bukan ?

Tanya: Icon-icon yang digunakan pada contoh program QSimpleApp cukup menarik. Bagaimana menghasilkan icon semacam ini ?
Jawab: Pada dasarnya, icon-icon ini "milik" KDE. Bila Anda menggunakan KDE 2, maka file-file icon bisa dilihat di subdirektori `share/icons/` dari direktori instalasi KDE (misalnya untuk SuSE, ini berarti `/opt/kde2`). Untuk contoh QSimpleApp, saya mengambil beberapa icon ukuran 16x16 locolor. Karena format aslinya adalah PNG, dengan menggunakan GIMP, icon-icon ini dikonversi dulu ke X Pixmap (alias XPM) dan dikumpulkan menjadi satu dalam sebuah file header.

Tutorial Qt (bagian 3)

Qt bukanlah penyedia widget semata. Qt lebih dari sekedar toolkit untuk mengkonstruksi antarmuka grafis atau GUI. Kali ini akan dibahas bagaimana Qt memudahkan penanganan struktur data. Seperti biasa, tersedia pula program contoh untuk melukiskan secara lebih jelas bagaimana hal ini bisa diwujudkan.

Manipulasi String dengan QString

Dalam bahasa C dan C++, sebuah string direpresentasikan sebagai array dari `char` dengan karakter 0 atau null sebagai pertanda akhir string. Yang semacam ini sering kemudian disebut sebagai *null-terminated string*. Pustaka standar C juga menyediakan sejumlah fungsi untuk melakukan manipulasi string, bisa dilihat di file header `string.h`.

Problem utama penggunaan string sebagai array char adalah keterbatasan dalam fleksibilitas penanganan memori. Hal ini karena string sering dialokasikan secara dinamik, sebagaimana array, sehingga membutuhkan pula proses dealokasi memori. Bila tidak dilakukan dengan tepat, alokasi/dealokasi ini bisa menyebabkan si programer sakit kepala mengingat kesalahan sedikit saja bisa berakibat fatal: *segmentation fault* karena salah dealokasi atau boros penggunaan memori (*memory leak*) karena lupa melakukan dealokasi. Belum lagi masalah dereferensi pointer yang harus dilakukan dengan hati-hati.

Masalah lain untuk string muncul ketika program yang dikembangkan harus berurusan dengan karakter yang bukan latin, misalnya huruf-huruf Kanji dari Jepang. Mengingat string adalah array dari char dan char hanya 8 bit, maksimal kombinasi karakter yang ditampung adalah 256 karakter. Hampir pada semua kasus, ke-256 karakter ini akan diisi dengan karakter ASCII. Lantas, bagaimana dengan orang-orang Rusia atau Urdu yang juga tidak mau ketinggalan bermain program komputer ? Bagaimana mereka harus merepresentasikan semua huruf-huruf yang non-latin ?

Solusi dari masalah ini adalah penggunaan *Unicode*, yaitu sebuah standar untuk menyimpang karakter atau string secara lebih baik sehingga sanggup menampung semua aneka karakter dari semua bahasa-bahasa di dunia. Triknya sederhana: memperluas kemungkinan kombinasi karakter dari 256 menjadi 65536 dengan mengubah representasi satu karakter menjadi 16-bit (tidak lagi 8-bit). Lebih jauh tentang Unicode, berikut pemetaan karakter-karakternya, bisa dilihat di www.unicode.org.

Sayang sekali Unicode belum diadopsi sebagai standar untuk digunakan di bahasa C ataupun C++. Untuk mengatasi hal ini, Qt menyediakan kelas QString, sebuah kelas yang dapat dipergunakan untuk menyimpan dan mengolah string dan sudah mendukung Unicode. Kalau string adalah array dari char, maka QString adalah array dari QChar, sebuah kelas yang mewakili satu karakter Unicode. Kelas QString ini juga dilengkapi sekian puluh fungsi-fungsi yang sangat berfaedah untuk memanipulasi string dan pastinya akan menjadi 'teman baik' seorang programmer (Tidak percaya ? Sembari maupun setelah membaca tulisan ini silakan merujuk ke referensi Qt tentang QString, mudah dipahami bahwa fungsionalitas yang disediakan QString memang lengkap dan yang betul-betul dibutuhkan).

Bagaiamana membuat string dengan QString ? Mudah sekali. Bisa dilakukan misalnya sebagai berikut:

```
QString s = "Hello"
```

atau:

```
QString s("Hi there")
```

Karena QString bisa melakukan konversi secara otomatis, maka beralih dari string (atau `char*`) bisa dilakukan semudah:

```
char msg[] = "Guten Morgen !";
QString s = msg;
```

Sebaliknya, casting dari QString ke char* juga dimungkinkan. Lihat yang berikut ini.

```
QString s = "hi";
printf( "%s\n", (const char*) s );
```

ataupun:

```
cout << s;
```

Pun mengalihkan objek QString ke stream tidak akan menimbulkan masalah:

```
QString ask = "Who are you?";
cout << ask << endl;
```

Fungsi `uint QString::length` akan mengembalikan panjang string (mirip dengan `strlen`). Sebagai contoh, bila `str` adalah "Linux", maka `str.length()` menghasilkan 5.

Fungsi `bool QString::isNull()` dan `bool QString::isEmpty()` digunakan untuk memeriksa apakah string NULL dan apakah string tersebut kosong. Perhatikan di sini bahwa empty != null. Yang dimaksud dengan *empty string* adalah string yang tidak berisi apa-apa (`length()` akan menghasilkan 0). Sementara itu *null string* bisa dianalogikan dengan NULL pada pointer. Terdapat juga fungsi statik `QString::null` yang akan mengembalikan *null string*. Lazimnya, dua fungsi ini digunakan pada pemeriksaan kondisi tertentu. Potongan kode berikut mengilustrasikannya:

```
// do nothing if null
if( s.isNull() ) return;

// empty or not ?
if( s.isEmpty() ) cout << "string is empty" << endl;

cout <<Your string is << s << endl;
```

Catatan: pada kode di atas, alih-alih `s.isNull()`, bisa juga digunakan `s == QString::null`.

Bagaimana mencomot sebagian string ? Ada fungsi `QString QString::left( uint len )`, `QString QString::right( uint len )`, dan `QString QString::mid( uint index, uint len )` yang masing-masing akan menghasilkan sebuah string baru yang merupakan bagian tertentu dari string yang dipanggil. Contoh berikut menunjukkan bagaimana hasil pemanggilan ketiga fungsi ini:

```
QString s = "Linus Torvalds" ;
QString p = s.left( 4 ) ;       // Linu
QString q = s.right( 3 );       // lds
QString r = s.mid( 6, 3 );      // Tor
```

Kalau begitu, sekarang bagaimana menggabungkan string ? Dibandingkan pustaka standar C yang hanya menyediakan `strcat`, Qt memberikan fungsi `QString& QString::append( const QString& s )` dan `QString& QString::prepend( const QString& s )` yang akan membubuhkan string `s` berturut-turut ke akhir dan awal string.

```
QString name = "Maddog" ;
name.append( "Hall" );     // Maddog Hall
name.prepend( "John" );    // John Maddog Hall
```

Ada pula `QString& QString::insert( uint index, const QString& s )` yang akan menyisipkan string s pada posisi index. Lihat contoh ini:

```
QString jfk = "John Kennedy" ;
cout << jfk.insert( 5,"F. " ) << endl; // John F. Kennedy
```

Dokumentasi Qt yang rinci dan lengkap menerangkan lebih banyak lagi mengenai kelas QString dan QChar ini.

Program `pig.cpp` di bawah ini mengilustrasikan penggunaan kelas QString. Yang dilakukan program ini - lazim dikenal sebagai Pig Latin - adalah menerima sebuah kata dan mencetak versi kebalikannya, yakni bila huruf-huruf pada kata tersebut ditulis dari arah kanan ke kiri.

Berikut adalah listing programnya:

```
 1: // pig.cpp - Pig Latin
 2: #include <iostream.h>
 3: #include <qstring.h>
 4:
 5: int main( int argc, char **argv )
 6: {
 7:   if( argc < 2 ) return -1;
 8:
 9:   QString name = argv[1];
10:   QString rev;
11:
12:   for(int i = name.length() - 1; i >= 0; i--)
13:     rev += name[i];
14:
15:   cout << rev << endl;
16:
```

```
17:   return 0;
18: }
```

Listing 1: pig.cpp

Lakukan compile dengan perintah:

```
g++ -o pig pig.cpp -lqt -I/usr/lib/qt2/include -L/usr/lib/qt2/lib
```

Catatan: Tidak perlu Makefile karena programnya cukup sederhana. Sesuaikan `/usr/lib/qt2` dengan direktori instalasi Qt.

Sekarang, bila program ini dijalankan, misalnya dengan perintah `./pig Linux`, maka hasilnya `xuniL`. Silakan mencobanya sendiri.

Sebagai latihan, program ini bisa dimodifikasi agar tidak hanya menerima parameter pertama program saja. Dengan demikian, yang di-Pig-Latin bisa saja sebuah kalimat utuh, bukan hanya satu kata.

Program `pig.cpp` juga mencontohkan bagaimana membuat aplikasi yang berbasis Qt, tetapi tidak menggunakan GUI (sering juga disebut aplikasi *console*). Dapat dilihat bahwa hal ini bisa diwujudkan dengan gampang, cukup dengan tidak menggunakan kelas-kelas yang merupakan widget (seperti QApplication, QMainWindow, dan lain-lain).

Kawan

Mungkin bagi sebagian pembaca program Pig Latin di atas tidak begitu berdaya pikat. Karenanya marilah beranjak mengulas sebuah program contoh lainnya, yaitu sebuah aplikasi kecil untuk mendata nama, alamat e-mail, dan nomor telfon, yang dibaptis sebagai program *Kawan*. Seperti biasa, untuk sekedar menggoda, screenshot program ini (yang mudah-mudahan menarik) bisa dinikmati di Gambar 1.

Gambar 1. Screenshot program Kawan

Sedikit tentang berfungsinya program Kawan ini akan dijelaskan terlebih dahulu. Tujuan program adalah menyajikan data, dalam hal ini datanya berasal dari sebuah file bernama `kawan.dat`. Bisa dikatakan bahwa file ini akan bertindak sebagai basis data mini. Untuk mudahnya, file `kawan.dat` ini merupakan file teks ASCII biasa yang bisa disunting di sembarang editor. Tiap-tiap baris pada file ini akan menyimpan satu record. Karena dibutuhkan tiga field, yaitu nama, alamat e-mail, dan nomor telfon, maka untuk masing-masing baris, terdapat tiga string yang dipisahkan oleh karakter : (titik dua) sebagai pembatas field.

Jika sekiranya, deskripsi di atas membingungkan, di bawah ini adalah contoh sebuah file `kawan.dat`. File data ini juga yang digunakan untuk menghasilkan screenshot di atas.

```
Esther:funkfyflo00@hotmail.com:0175 6969224
Ailsa:ailsa_gini@hotmail.com:0160 98534128
Elin:elin77@web.de:0179 6761591
Lia:fidia17@yahoo.com:0162 4160266
Rochim:rochim_h@yahoo.de:0170 8459166
Nano:estananto@yahoo.de:0179 5024705
Koredianto:kore76@yahoo.com:0175 9265806
Nova:nuzera@yahoo.com:0179 5158845
Sindy:sdewiana@yahoo.com:0179 3684614
```

Listing 2. Contoh file kawan.dat

Lebih dari itu, program Kawan juga diinginkan punya fasilitas untuk mengedit, menghapus, serta menambah data baru yang dapat diakses melalui menu program.

Secara fungsional dapat disebutkan bahwa minimal program Kawan ini harus bisa:

- mengambil data dari file
- menyimpan data ke file
- mendaftar dan menyajikan data
- menghapus sebuah record
- mengedit record
- menambah record baru

Mengambil data dari file bukan pekerjaan yang susah. Prosedurnya pun relatif sederhana: buka file `kawan.dat`, baca baris per baris, untuk tiap barisnya pisahkan ketiga field, tambahkan sebagai record baru. Adapun penyimpanan data ke file adalah proses kebalikannya: untuk tiap record buat sebuah string dan simpan sebagai satu baris ke file data.

Untuk menyajikan data, digunakan widget yang bernama QListView. Tiap-tiap record akan berupa satu QListViewItem untuk widget QListView ini. Pada screenshot yang ditunjukkan sebelumnya, widget QListView adalah widget utama yang berada di tengah window.

Guna mengimplementasikan program Kawan ini, masih digunakan pendekatan yang sama sebagaimana telah dikupas di bagian pertama dan kedua seri tulisan ini. Akan terdapat tiga buah source code, masing-masing untuk program utama (`main.cpp`), definisi kelas Kawan (`kawan.h`), serta implementasi kelas Kawan (`kawan.cpp`). Juga dibutuhkan sebuah `Makefile` untuk memudahkan proses kompilasi.

Di bawah ini adalah listing program untuk file `kawan.h`.

```
 1: // kawan.h
 2:
 3: #ifndef __KAWAN_H
 4: #define __KAWAN_H
 5:
 6: class QString;
 7: class QListView;
 8: class QListViewItem;
 9: #include <qlist.h>
10: #include <qmainwindow.h>
11:
12: class Kawan: public QMainWindow
13: {
14:     Q_OBJECT
15:
16: public:
17:     Kawan();
18:
19: protected:
20:     void initMenu();
21:     void initToolbar();
22:     void loadData();
23:     void saveData();
24:
25: private slots:
26:
27:     void slotEntryAdd();
28:     void slotEntryEdit();
29:     void slotEntryDelete();
30:     void slotAbout();

31:
32: private:
33:
34:     QListView *list;
35:     QList<QListViewItem> entries;
36:
37: };
38:
39: #endif
```

Listing 3. kawan.h

File `kawan.h` hanya berisi definisi kelas Kawan. Tidak ada sesuatu yang istimewa di sini. Anda juga bisa melihat bahwa terdapat tiga slot bernama `slotEntryAdd()`, `slotEntryEdit()`, dan `slotEntryDelete()` yang masing-masing digunakan untuk menambah, mengedit, serta menghapus record. Sebagaimana sudah disebutkan bahwa

QListView digunakan untuk menyajikan daftar record, yang pada file header di atas diletakkan dalam pointer `list` (lihat baris 34). Yang baru di sini adalah variabel bernama `entries` yang merupakan sebuah objek dari kelas QList. Patut dijelaskan di sini bahwa QList adalah sebuah *template class* yang berarti bahwa harus diberikan kelas yang akan menjadi item dari QList tersebut. Untuk kasus ini, item tersebut adalah QListViewItem.

Bagaimana implementasi kelas Kawan ? Lihatlah terlebih dahulu file `kawan.cpp` berikut ini:

```
  1: // kawan.cpp - Simple contact manager
  2:
  3: #include <qapplication.h>
  4: #include <qfile.h>
  5: #include <qlistview.h>
  6: #include <qmenubar.h>
  7: #include <qmessagebox.h>
  8: #include <qstatusbar.h>
  9: #include <qtextstream.h>
 10:
 11: #include "kawan.moc"
 12:
 13: const char* datafile = "kawan.dat";
 14:
 15: Kawan::Kawan() : QMainWindow( 0, "Kawan", WDestructiveClose )
 16: {
 17:   entries.setAutoDelete( TRUE );
 18:
 19:   initMenu();
 20:
 21:   list = new QListView( this );
 22:   list->addColumn( "Name" );
 23:   list->addColumn( "E-mail" );
 24:   list->addColumn( "Phone" );
 25:   list->setAllColumnsShowFocus( TRUE );
 26:   setCentralWidget( list );
 27:
 28:   loadData();
 29:
 30:   resize( 400, 250 );
 31: }
 32:
 33: void Kawan::loadData()
 34: {
 35:   QFile f( datafile );
 36:   if ( !f.open( IO_ReadOnly ) )
 37:     return;
 38:
 39:   entries.clear();
 40:
 41:   QTextStream stream( &f );
 42:   while ( !stream.eof() )
 43:     {
 44:       QStringList s = QStringList::split( ":", stream.readLine(),
TRUE );
 45:       QString name = s[0].stripWhiteSpace();
 46:       QString email = s[1].stripWhiteSpace();
 47:       QString phone = s[2].stripWhiteSpace();
```

```
 48:       QListViewItem* e = new QListViewItem( list, name, email, phone
);
 49:       entries.append( e );
 50:     }
 51:
 52:   f.close();
 53:
 54:   statusBar()->message( QString("%1 entries").arg( entries.count()
), 2000 );
 55: }
 56:
 57: void Kawan::saveData()
 58: {
 59:   QFile f( datafile );
 60:   if ( !f.open( IO_WriteOnly ) )
 61:     return;
 62:
 63:   QTextStream stream( &f );
 64:
 65:   QListIterator<QListViewItem> it( entries );
 66:   for(; it.current(); ++it )
 67:     {
 68:       QListViewItem* e = it.current();
 69:       QString s = QString( "%1:%2:%3" ).arg( e->text(0) ).
 70:    arg( e->text(1) ).arg( e->text(2) );
 71:       stream << s << endl;
 72:     }
 73:
 74:   f.close();
 75: }
 76:
 77: void Kawan::initMenu()
 78: {
 79:     QPopupMenu * entry = new QPopupMenu( this );
 80:     menuBar()->insertItem( "&Entry", entry );
 81:     entry->insertItem( "Add", this, SLOT( slotEntryAdd() ) );
 82:     entry->insertItem( "Delete", this, SLOT( slotEntryDelete() ) );
 83:     entry->insertItem( "Edit", this, SLOT( slotEntryEdit() ) );
 84:     entry->insertSeparator();
 85:     entry->insertItem( "Close", this, SLOT( close() ) );
 86:
 87:     QPopupMenu * help = new QPopupMenu( this );
 88:     menuBar()->insertSeparator();
 89:     menuBar()->insertItem( "&Help", help );
 90:     help->insertItem( "&About...", this, SLOT( slotAbout())  );
 91: }
 92:
 93: void Kawan::slotEntryAdd()
 94: {
 95: }
 96:
 97: void Kawan::slotEntryEdit()
 98: {
 99: }
100:
101: void Kawan::slotEntryDelete()
102: {
103:   QListViewItem* e = list->currentItem();
104:   if( e )
```

```
105:     {
106:       list->takeItem( e );
107:       saveData();
108:       statusBar()->message( "Deleted", 1000 );
109:     }
110: }
111:
112: void Kawan::slotAbout()
113: {
114:   QMessageBox::about( this, "Kawan",
115:                 "<b>Kawan 0.1</b>"
116:                 "<p>Simple contact manager</p>"
117:                 "<p>Programmed by Ariya Hidayat<br>"
118:                 "ariya@infolinux.co.id</p>" );
119: }
```

Listing 4. kawan.cpp

Ada beberapa hal menarik untuk kelas Kawan ini.

Di awal sekali, bisa dilihat bahwa "kawan.dat" ditetapkan sebagai nama file data untuk program ini.

Pada konstruktor kelas Kawan, bisa dilihat bagaimana inisialisasi `list` sebagai sebuah widget `QListView`. QListView bisa menampung sekian kolom, masing-masing ditambahkan dengan `QListView::addColumn`. Perhatikan pula terjadi AutoDelete pada `entries` (yang merupakan QList). Dengan `QList::setAutoDelete`, bisa diatur apakah AutoDelete akan diaktifkan atau tidak. Bila aktif, hal ini berarti apa saja yang ditambahkan ke entries akan dihapus secara otomatik dari memori manakala entries juga menjadi invalid.

Fungsi `Kawan::loadData` bertugas membaca record-record dari file data. Di sini digunakan QFile untuk mengakses file (low-level) dan QTextStream untuk bekerja dengan isi dari file tersebut (high-level). Mula-mula tiap baris dari file data diletakkan dalam sebuah string dan kemudian dipisahkan berdasarkan karakter : (titik dua). Dengan Qt, hal ini menjadi mudah karena tersedia `QStringList` yang mempunyai fungsi `QStringList::split` (yang tidak asing dengan Perl atau Python bisa melihat kesamaannya di sini). Sebetulnya QStringList tidak lain adalah list dari beberapa QString. Dengan hanya sekitar 20 baris program, fungsionalitas untuk mengambil record dari file data sudah terwujudkan.

Fungsi `Kawan::saveData` adalah kebalikan dari `loadData`, yaitu untuk menyimpan record ke file data. Tidak lebih mudah dan juga tidak lebih sulit, fungsi `saveData` ini cukup sederhana: proses record satu demi satu, untuk tiap recordnya susun sebuah string yang kemudian ditulis ke file. Lagi-lagi akses ke file menggunakan QFile dan QTextStream. Loop untuk membaca masing-masing item dalam variabel entries dilakukan dengan iterator, yakni `QListIterator`. Pembaca yang paham dengan penggunaan STL (Standard Template Library) tentunya tidak akan asing lagi dengan konsep iterator ini. Ringkasnya, iterator ini adalah objek yang bisa digunakan untuk

bergerak mengakses entri-entri pada sebuah objek lain. Karena `entries` berasal dari kelas QList, maka iterator yang digunakan adalah QListIterator.

Mengkomposisikan string di Qt jauh lebih gampang dibandingkan dengan C atau C++ biasa. Mengapa ? Hal ini tidak lain karena tersedianya `QString::arg` yang terbilang *powerful* untuk urusan ini. Fungsi akan menghasilkan sebuah string baru (masing sebagai QString tentunya) yang sama dengan format yang diberikan tetapi dengan mengganti %d dengan argumen yang diberikan. Di sini d adalah sebuah integer. Fungsi `arg` hanya akan mensubstitusi integer d yang terkecil. Dengan demikian, bila `s` berisi "Val %1", maka `s.arg( 0.707 )` akan menghasilkan " Val 0.707". Cermati juga bahwa dengan keindahan polymorphism di C++, `arg` dapat menerima apa saja: integer, double, char, QString, dan lain-lain.

Lazimnya, fungsi `arg` ini diatur secara seri (atau *cascade*) untuk menghasilkan sebuah string yang terformat rapi. Ilustrasi kecil, perhatikan potongan kode di bawah:

```
QString name = "Joe";
double pi = 3.14;
QString formatstr = "Name is %1 and PI is %2";
cout << formatstr.arg( name ).arg( pi ) << endl;
```

Hasilnya adalah:

```
Name is Joe and PI is 3.14
```

Di sini %1 akan digantikan dengan `name` dan %2 digantikan dengan `pi`. Alokasi dan dealokasi memori diatur secara otomatis. Lebih cepat dan gampang dibandingkan `sprintf()`, bukan ?

Kembali ke program Kawan, `QString::arg` dimanfaatkan untuk menghasilkan string untuk menampung sebuah record sesuai dengan format file data `kawan.h` (baris 68-71).

Fungsi `Kawan::initMenu` akan mengkonstruksi menu untuk program contoh ini. Saat sekarang, menu utamanya hanya dua: Entry dan Help. Untuk menu Entry, ada empat submenu: Add, Delete, Edit, dan Close yang masing-masing dihubungkan dengan empat slot berturut-turut: `slotEntryAdd`, `slotEntryDelete`, `slotEntryEdit`, dan `close`. Kecuali yang terakhir, tiga yang pertama adalah slot yang khusus dimiliki Kawan. Sementara itu, menu Help hanya berisi submenu About (terhubung ke slot `slotAbout`).

Sisa file adalah implementasi keempat slot yang baru saja disebutkan (slot `close` sudah diwarisi dari QMainWindow). Perhatikan bahwa baik `slotAdd` dan `slotEdit` belum berisi apa-apa, keduanya akan dibahas di bagian empat seri tulisan ini. Dus, program Kawan ini sebenarnya juga kehilangan dua fungsionalitas: menambah dan menyunting record.

Slot untuk menghapus record, `Kawan::slotEntryDelete`, relatif pendek. Slot ini memanfaatkan fungsi `QList::takeItem` untuk menghapus sebuah item dari QList. Item yang mana yang dihapus sendiri ditentukan dari `QListView::currentItem`. Sebuah

blok if diperlukan di sini karena `QListView::currentItem` bisa mengembalikan NULL jika memang tidak ada item yang disorot oleh user. Setelah sebuah record dihapus hal ini berarti data mengalami perubahan. Oleh karenanya `saveData` dipanggil untuk menyimpan data yang baru ke file.

Sebetulnya program sekecil Kawan tidak membutuhkan About Box. Fungsi `Kawan::slotAbout` sendiri sengaja ditambahkan untuk mengilustrasikan salah satu fitur Qt, yaitu kemampuannya menampilkan *rich-text*. Yang dimaksud sebagai rich-text di sini adalah teks yang tidak hanya sekedar susunan karakter biasa, tetapi lebih ke teks yang bisa diformat, apakah itu *bold*, *italics*, dan sebagainya.

Bila diamati, terlihat bahwa `QMessageBox::about` dipanggil dengan string yang mirip dengan kode-kode HTML. Mudah ditebak bahwa sebuah teks yang akan diformat sebagai rich-text diatur dengan menempatkan tag-tag HTML di dalam teks tersebut. Hanya ada dua tag yang digunakan di sini: yaitu <b> untuk menghasilkan cetak tebal (*bold*) dan <p> untuk berganti baris (*paragraph*). Gambar 2 menunjukkan bagaimana hasil About Box dengan menggunakan rich-text.

Gambar 2. About Box

Kelas Kawan yang sudah lengkap didefinisikan dan diimplementasikan masih belum cukup untuk membangun sebuah aplikasi Qt yang utuh. Dibutuhkan rutin utama yang menghasilkan QApplication yang kemudian menggunakan objek dari kelas Kawan sebagai window utama. Hal ini diwujudkan di file source code yang ketiga, yaitu `main.cpp`. Adapun listing file tersebut adalah:

```
 1: // main.cpp
 2:
 3: #include <qapplication.h>
 4: #include "kawan.h"
 5:
 6: int main( int argc, char ** argv )
 7: {
 8:   QApplication a( argc, argv );
 9:
10:   Kawan *w = new Kawan();
11:   w->show();
12:
13:   a.connect( &a, SIGNAL(lastWindowClosed()), &a, SLOT(quit()) );
14:
15:   return a.exec();
16: }
```

Listing 5. main.cpp

Untuk memudahkan proses kompilasi, digunakan Makefile berikut ini:

```
1: QTDIR     = /usr/lib/qt2
```

```
 2: QTLIB     = $(QTDIR)/lib
 3: QTINCLUDE = $(QTDIR)/include
 4: QTMOC     = $(QTDIR)/bin/moc
 5:
 6: TARGET    = kawan
 7: OBJECTS   = kawan.o main.o
 8: MOCS      = kawan.moc
 9:
10: CC        = g++
11: CFLAGS    = -Wall -O2
12: LIBS      = -lqt
13:
14: $(TARGET): $(MOCS) $(OBJECTS)
15:    $(CC) $(LFLAGS) -o $(TARGET) $(OBJECTS) $(LIBS) -L$(QTLIB)
-I$(QTINCLUDE)
16:
17: %.o: %.cpp
18:    $(CC) $(CFLAGS) -I$(QTINCLUDE) -c $< -o $@
19:
20: %.moc: %.h
21:    $(QTMOC) $< -o $@
22:
23: clean:
24:    rm $(OBJECTS) $(TARGET) $(MOCS)
```

Listing 6. Makefile

Kalau Anda cukup teliti, bisa dilihat bahwa Makefile ini nyaris sama seperti yang sudah-sudah (lihat bagian 1 dan 2 seri tulisan ini). Yang berbeda hanya baris 6, 7, dan 8 untuk menyesuaikan dengan nama programnya. Sekedar mengingatkan lagi, jangan lupa menyesuaikan QTDIR dengan direktori instalasi Qt pada sistem Anda.

Sekarang proses kompilasi bisa dilakukan dengan mudah, cukup jalankan perintah `make`. Jika tidak ada sesuatu yang salah, maka program `kawan` sudah siap untuk digunakan. Jangan lupa juga untuk membuat file `kawan.dat` yang dibutuhkan untuk program ini.

Bonus untuk program Kawan, daftar yang disajikan bisa disortir dengan mudah. Tidak lain tidak bukan yakni dengan mengklik pada judul kolomnya, apakah itu untuk *Name*, *E-mail*, maupun *Phone*. Anda bisa mencobanya sendiri.

Sebagai latihan, Anda bisa menambahkan toolbar untuk program Kawan ini. Tentunya hal ini tidak terlalu susah mengingat hal-hal yang berkenaan dengan toolbar sudah dikupas di bagian kedua seri tulisan ini. Untuk icon (atau pixmap) pada toolbar ini, Anda bisa mengambilnya dari mana saja, sepanjang ukurannya tepat.

Tanya Jawab

Tanya:
Jawab:

Tanya: Bagaimana menghilangkan spasi kosong yang tidak perlu pada string ?
Jawab: Gunakan fungsi `QString` `QString::stripWhiteSpace()`. Misalnya `QString s = " some text "`, maka `s.stringWhiteSpace()` menghasilkan string baru "some text".

Tanya: Saya biasa memformat string dengan `sprintf()`. Dengan QString, bagaimana hal ini bisa dilakukan ?
Jawab: Untunglah `sprintf()` juga merupakan metoda dalam kelas QString. Hal ini berarti mirip dengan `sprintf()` biasa, Anda dapat melakukannya ke dalam sebuah string dari QString. Sebagai ilustrasi, coba periksa hasil dari `QString s; s.sprintf( "%g", M_PI );` (jangan lupa meng-include-kan `math.h`). Alternatif lain adalah menggunakan metoda `arg()`, sebagaimana ditunjukkan pada program contoh.

Tanya: Untuk kelas-kelas yang saya buat sendiri dan tidak berasal dari Qt, apakah harus digunakan QString untuk menangani string ? Bukankan char* saja sudah cukup ?
Jawab: Bila Anda yakin tidak akan menggunakan Unicode, hal ini bukan masalah. Namun demikian, mengingat begitu mudahnya operasi string dilakukan dengan kelas QString, tidak ada salahnya untuk menggunakan kelas QString. Anda akan diuntungkan juga karena tidak perlu melacak bug yang diakibatkan kesalahan alokasi dan dealokasi memori atau dereferensi pointer yang tidak tepat.

Tanya: Operasi manipulasi karakter dan string dengan menggunakan kelas QString bukankan tidak efisien dan menurunkan kecepatan program ?
Jawab: Kelas QString dirancang agar tidak melakukan *copy* yang tidak perlu, yaitu dengan menerapkan *shallow copy*. Hal ini berarti dua string yang sama tidak akan memiliki dua instance yang berbeda. Secara tidak langsung, hal ini mengurangi sekali ketidakefisienan QString sehingga tidak mempunyai pengaruh yang signifikan terhadap kecepatan eksekusi program. Bila dibandingkan dengan fasilitas yang diberikan, menggunakan QString merupakan kompromi yang baik antara kemudahan dan efisiensi.

Tanya: Jika saya menggunakan QList, sering muncul masalah *Segmentation fault*.
Jawab: Kemungkinan besar Anda mencoba menghapus objek yang sudah tidak valid lagi, misalnya dengan operator `delete`. Bila objek ini telah disisipkan ke dalam sebuah QList dan Anda sudah mengaktifkan `setAutoDelete( TRUE )`, maka tidak perlu lagi menghapus objek ini secara manual. AutoDelete berarti bahwa objek yang terdapat pada sebuah QList akan dihapus bersamaan ketika QList tersebut juga menghilang dari memory.

Tanya: Dengan `QList::append`, sebuah item disisipkan di akhir. Bagaimana untuk menyisipkannya di awal sehingga item ini menjadi item yang pertama dalam sebuah QList ?
Jawab: Mudah saja. Gunakan `QList::prepend`.

Tanya: Apakah QList dan QListView berhubungan ?
Jawab: Tidak. QList adalah kelas yang digunakan untuk menampung sejumlah entri, mirip seperti sebuah list biasa. QListView adalah widget yang lumrahnya digunakan untuk mendaftar sejumlah data. Keduanya tidak saling tergantung dan bisa digunakan terpisah.

Tanya: Saya perhatikan QList ini serupa dengan kelas-kelas yang ada pada STL (Standard Template Library). Adakah kelas Qt yang lain yang mirip penggunannya dengan STL ?
Jawab: Ada banyak. Selain QList terdapat juga QMap, QArray, QDict, QStack, dan lain-lain. Silakan merujuk ke manual Qt untuk lebih jelasnya. Kelas-kelas Qt yang ini seperti memang serupa dengan STL sehingga sering dinamakan juga sebagai QTL (Qt Template Library).

Tanya: Jika saya hitung-hitung, untuk menghasilkan program semacam Kawan dibutuhkan sekitar 180 kode C++ (mencakup semua file source code dan juga Makefile-nya). Tidakkah ini terlalu banyak untuk program sesederhana itu ?
Jawab: Tidak juga. Anda tentu bisa mengimplementasikan fungsionalitasnya yang sama dan tidak menggunakan Qt, tetapi dengan library lainnya semisal Xlib, Motif, maupun GTK. Bisa dijamin bahwa program yang menggunakan Qt (sebagaimana Kawan di atas) lebih pendek, ringkas, dan mudah dipahami.

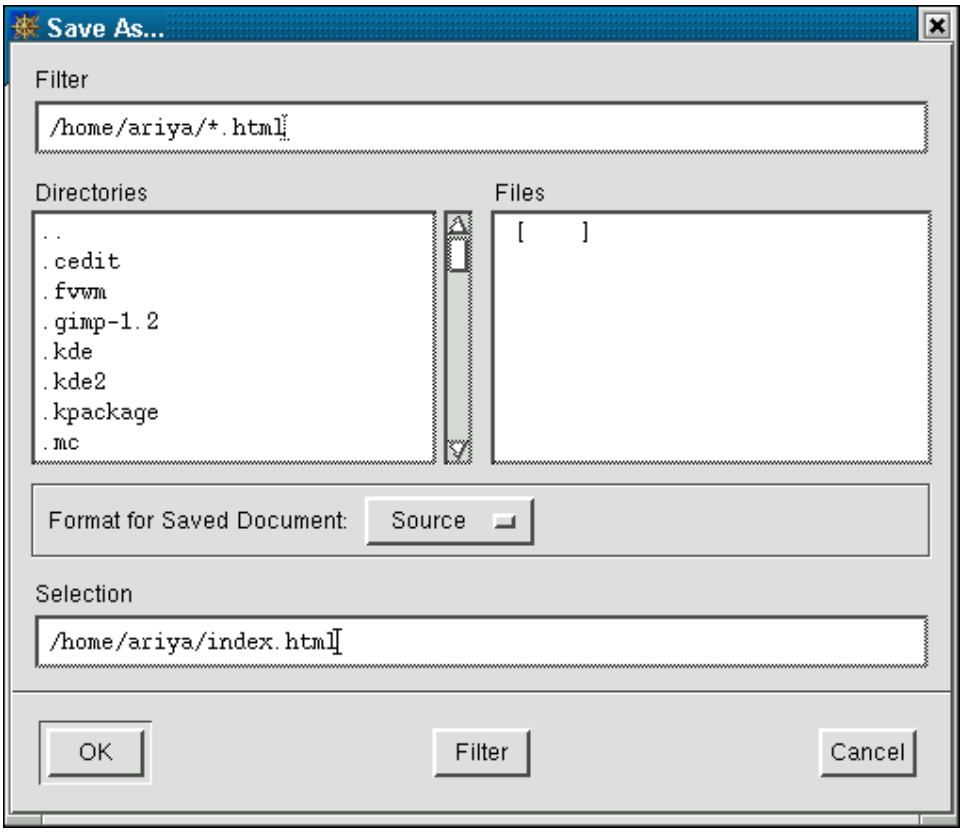
Tanya: Saya tertarik dengan screenshot program Kawan. Sepertinya windownya cukup unik. Bagaimana menghasilkan window semacam ini ?
Jawab: Yang ditampilkan sebagai screenshot sebetulnya adalah dekorasi IceWM yang bernama MenschMaschine. Jadi, tidak ada yang aneh dengan programnya. Anda tinggal menggunakan IceWM sebagai window manager, mengaktifkan MenschMaschine, dan semua program akan memiliki dekorasi window yang dimaksud. Cara lain (yaitu yang dilakukan untuk menghasilkan screenshot di atas) adalah dengan menggunakan dekorasi IceWM ini di KDE 2.2.

Tutorial Qt (bagian 4)

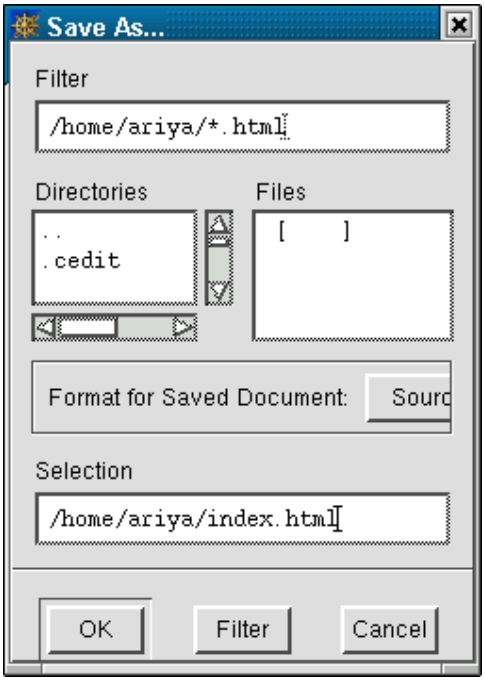
Programer yang kawakan mestilah sudah paham bagaimana rumitnya mengatur tata letak dari elemen-elemen *user-interface*. Untuk mengatasi hal ini, Qt menggunakan pendekatan seperti yang diterapkan di Java dalam penataan widget-widget, yakni dengan sebuah kelas khusus *QLayout* untuk melakukan layout secara mudah dan fleksibel.

Tata Letak: Pengaturan Geometri Widget

Sebelum melihat bagaimana user interface dapat dibangun dengan kelas-kelas widget yang disediakan oleh Qt, ada baiknya disentuh sedikit mengenai *geometry management*. Bagi yang pernah memprogram di Windows, misalnya dengan Visual Basic atau Delphi, jelas bahwa menyusun elemen user interface, seperti button, checkbox, list, dan lain-lain dilakukan dengan meletakkan elemen- elemen tersebut pada posisi tertentu. Namun demikian, buat yang sedikit punya pengalaman dengan Motif ataupun Java, elemen-elemen user interface tidak disusun pada posisi yang tepat. Hal ini berarti lokasi relatif sebuah button dapat berubah, manakala window di mana button tersebut berada mengalami perubahan ukuran. Perhatikan bagaimana kotak dialog (misalnya Save File) dari program yang berbasis Motif - misalnya Netscape 4.x - dapat 'menyesuaikan diri' ketika di-resize, sebagaimana diperbandingkan di Gambar 1 dan 2 berikut ini.



Gambar 1. Kotak Dialog Save File di Netscape



Gambar 2. Kotak Dialog yang diubah ukurannya

Bagaimana melakukan ini di Qt? Beruntunglah bahwa Qt menyediakan kelas QLayout yang memang berguna untuk menata widget-widget dengan cara yang sederhana tetapi cukup *powerful*. Pada prakteknya, yang dipakai bukanlah QLayout tetapi kelas turunan dari QLayout, yaitu QGridLayout dan QBoxLayout. Yang belakangan disebut malahan punya turunan lagi: QHBoxLayout dan QVBoxLayout. Semuanya akan dikupas satu per satu di bawah ini.

Kelas QHBoxLayout dipergunakan untuk menyusun widget-widget secara mendatar atau horizontal, sebagaimana diilustrasikan pada Gambar 3 di bawah ini:

Gambar 3. Hasil dari QHBoxLayout

Fragmen kode yang digunakan untuk menghasilkan tampilan di atas ditunjukkan di bawah ini:

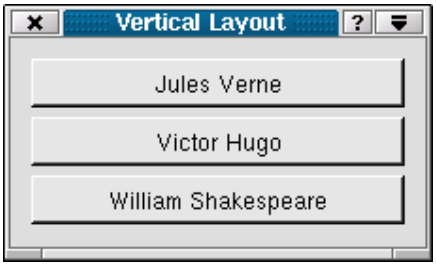
```
QBoxLayout *layout = new QHBoxLayout( this );
layout->setMargin( 10 );
layout->setSpacing( 3 );

layout->addWidget( new QPushButton ( "Mars", this ) );
layout->addWidget( new QPushButton ( "Jupiter", this ) );
layout->addWidget( new QPushButton ( "Saturnus", this ) );
```

```
layout->addWidget( new QPushButton ( "Uranus", this ) );
layout->addWidget( new QPushButton ( "Neptunus", this ) );
```

Jadi, bisa dilihat bahwa tekniknya adalah membuat objek dari QHBoxLayout dan kemudian menambahkan widget-widgetnya ke dalam objek ini dengan menggunakan fungsi `QHBoxLayout::addWidget()`. Widget yang pertama kali dimasukkan ke QHBoxLayout akan berada di posisi paling kiri (lihat button *Mars* pada contoh di atas).

Dua fungsi yang tercantum di potongan kode di atas adalah `QHBoxLayout:setMargin()` dan `QHBoxLayout::setSpacing()`. Fungsi yang pertama adalah untuk menentukan margin, yaitu jumlah pixel yang mengelilingi objek layout ini. Dalam contoh kasus ini, Anda bisa membayangkan bahwa sekelompok button ini - dari *Mars* hingga *Neptunus* dikurung oleh pagar yang tidak nampak, yang berukuran 10 pixel. Adapun `setSpacing` menentukan ruang kosong (juga dalam pixel) sebagai jarak antara satu widget dengan widget lainnya. Jelas bahwa baik margin maupun spacing harus diatur agar menghasilkan susunan yang baik.

Bilamana widget-widgetnya diinginkan untuk disusun secara tegak alias vertikal, maka layout yang cocok untuk ini adalah QVBoxLayout (Gambar 4).



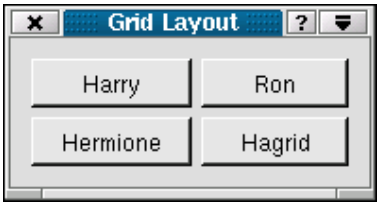
Gambar 4. Hasil dari QVBoxLayout

Menghasilkan yang seperti screenshot di atas masih mirip dengan contoh sebelumnya, perhatikan yang berikut ini:

```
QBoxLayout *layout = new QVBoxLayout( this );
layout->setMargin( 10 );
layout->setSpacing( 3 );

layout->addWidget( new QPushButton ( "Jules Verne", this ) );
layout->addWidget( new QPushButton ( "Victor Hugo", this ) );
layout->addWidget( new QPushButton ( "William Shakespeare", this ) );
```

Sama saja, bukan ? Hanya di sini digunakan QVBoxLayout, bukan QHBoxLayout. Widget pertama dari QVBoxLayout akan diletakkan paling atas, sebagaimana yang terjadi dengan *Jules Verne*.

Kelas layout yang terakhir, yaitu QGridLayout dipergunakan ketika widget-widgetnya seakan-akan disusun dalam sebuah matriks atau tabel. Karenanya, juga harus ditentukan terlebih dahulu jumlah baris dan jumlah kolom dari grid yang harus dihasilkan. Screenshot di bawah ini mencontohkan penggunaan QGridLayout.



Gambar 5. Hasil dari QGridLayout

```
QGridLayout *layout = new QGridLayout( this, 2, 2 );
layout->setMargin( 10 );
layout->setSpacing( 3 );

layout->addWidget( new QPushButton ( "Harry", this ), 0, 0 );
layout->addWidget( new QPushButton ( "Ron", this ), 0, 1 );
layout->addWidget( new QPushButton ( "Hermione", this ), 1, 0 );
layout->addWidget( new QPushButton ( "Hagrid", this ), 1, 1 );
```

Sebagai catatan, lepas dari jenis layout mana yang akan digunakan, file header yang di-include-kan cukup `qlayout.h`. Jadi tidak ada file header terpisah untuk masing-masing kelas penting seperti lumrahnya tradisi Qt.

Program Kawan: Versi 0.2

Sesuai yang dijanjikan di bagian ketiga seri tulisan ini, tibalah masanya untuk membahas mengenai program Kawan versi yang lebih baru - sebut saja versi 0.2. Seperti bisa dilihat di bagian ketiga, program Kawan versi sebelumnya masih belum mempunyai fasilitas menambah dan mengedit data. Kedua hal inilah yang akan dilengkapi sekarang.

Per desain, penambahan maupun penyuntingan data akan menyebabkan sebuah kotak dialog baru ditampilkan ke user, sudah beserta field-field yang akan ditambahkan atau diedit oleh user tersebut. Karenanya, akan dibutuhkan dua buah kelas baru: EditDialog dan AddDialog. Bisa dilihat bahwa kedua dialog ini akan mirip secara tampilan dan hanya berbeda sedikit secara fungsi. Oleh karenanya, diputuskan untuk membuat kelas AddDialog sebagai turunan dari kelas EditDialog. Kedua-duanya akan diletakkan di sebuah file terpisah dan tidak digabungkan ke program utama Kawan. Dengan demikian, walhasil akan ada dua tambahan file baru: `edit.h` untuk deklarasi EditDialog dan AddDialog serta `edit.cpp` untuk implementasinya.

Mula-mula, marilah kita lihat file header `kawan.h` yang sama sekali tidak mengalami perubahan dibandingkan versi 0.1.

```
 1: // kawan.h
 2:
 3: #ifndef __KAWAN_H
 4: #define __KAWAN_H
 5:
 6: class QString;
 7: class QListView;
 8: class QListViewItem;
 9: #include <qlist.h>
10: #include <qmainwindow.h>
```

```
11:
12: class Kawan: public QMainWindow
13: {
14:     Q_OBJECT
15:
16: public:
17:     Kawan();
18:
19: protected:
20:     void initMenu();
21:     void initToolbar();
22:     void loadData();
23:     void saveData();
24:
25: private slots:
26:
27:     void slotEntryAdd();
28:     void slotEntryEdit();
29:     void slotEntryDelete();
30:     void slotAbout();
31:
32: private:
33:
34:     QListView *list;
35:     QList<QListViewItem> entries;
36:
37: };
38:
39: #endif
```

Listing 1. kawan.h

Bagaimanakah membuat sebuah dialog untuk digunakan sebagai EditDialog dan AddDialog ? Dengan kekayaan API dari Qt serta kemudahan penggunakan kelas di C++, maka proses pembangunan sebuah kotak dialog cukup sederhana: turunkan saja dari kelas `QDialog`. Untuk jelasnya, perhatikan listing file `edit.h` yang mendeklarasikan kelas EditDialog berikut ini.

```
 1: // edit.h
 2:
 3: #ifndef __EDIT_H
 4: #define __EDIT_H
 5:
 6: #include <qdialog.h>
 7: class QLineEdit;
 8: class QPushButton;
 9: class QString;
10:
11: class EditDialog: public QDialog
12: {
13:   Q_OBJECT
14:
15:   public:
16:     EditDialog( QWidget* parent, const QString& _name,
17:          const QString& _email, const QString& _phone );
18:   private:
19:     QPushButton *button_dismiss;
```

```
20:     QPushButton *button_cancel;
21:     QLineEdit *edit_name;
22:     QLineEdit *edit_email;
23:     QLineEdit *edit_phone;
24:
25:   private slots:
26:     void slotDismiss();
27:     void slotCancel();
28:
29:   public:
30:     QString name;
31:     QString email;
32:     QString phone;
33: };
34:
35: class AddDialog: public EditDialog
36: {
37:   public:
38:     AddDialog( QWidget* parent );
39: };
40:
41: #endif
```

Listing 2. edit.h

Juga bisa disaksikan, seperti sudah disinggung beberapa menit yang lalu, kelas AddDialog yang digunakan untuk menambah entri baru akan dijadikan kelas turunan dari EditDialog.

Rancangan dialognya sendiri adalah bahwa terdapat tiga buah kotak isian, masing-masing untuk nama, e-mail, serta nomor telfon. Ini bersesuaian dengan field-field yang ada di program Kawan. Terdapat pula dua buat tombol perintah, masing-masing untuk mengaktifkan penyuntingan (*Dismiss*) dan membatalkannya (*Cancel*). Jenis widget yang akan dimanfaatkan hanya dua, yaitu `QPushButton` dan `QLineEdit`. Kalau Anda cermati, ada dua buah slot pada EditDialog yakni `slotDismiss` dan `slotCancel`. Masing-masing akan terhubung ke tombol perintah (button) yang bersesuaian.

Berikut bisa diperhatikan implementasi EditDialog dan AddDialog pada file `edit.cpp`.

```
 1: // edit.cpp
 2:
 3: #include <qdialog.h>
 4: #include <qlabel.h>
 5: #include <qlayout.h>
 6: #include <qlineedit.h>
 7: #include <qpushbutton.h>
 8:
 9: #include "edit.moc"
10:
11: EditDialog::EditDialog( QWidget* parent, const QString& _name,
12:                    const QString& _email, const QString& _phone ):
13:   QDialog( parent, "EditDialog", TRUE )
14: {
15:   QBoxLayout *topLayout = new QVBoxLayout( this, 3 );
16:
```

```
17:   setCaption( "Edit" );
18:
19:   // edit fields
20:   edit_name = new QLineEdit( this );
21:   edit_name->setText( _name );
22:   edit_email = new QLineEdit( this );
23:   edit_email->setText( _email );
24:   edit_phone = new QLineEdit( this );
25:   edit_phone->setText( _phone );
26:
27:   // arrange the fields
28:   QGridLayout *grid = new QGridLayout( topLayout, 3, 3 );
29:   grid->setSpacing( 4 );
30:   grid->setMargin( 5 );
31:   grid->addWidget( new QLabel( "Name", this ), 0, 0 );
32:   grid->addWidget( new QLabel( "E-mail", this ), 1, 0 );
33:   grid->addWidget( new QLabel( "Phone", this ), 2, 0 );
34:   grid->addWidget( edit_name, 0, 1 );
35:   grid->addWidget( edit_email, 1, 1 );
36:   grid->addWidget( edit_phone, 2, 1 );
37:
38:   // flexible spacer
39:   QWidget *spacer = new QWidget( this );
40:   spacer->setSizePolicy ( QSizePolicy( QSizePolicy::Expanding,
41:                                        QSizePolicy::Expanding ));
42:   topLayout->addWidget( spacer );
43:
44:   // command buttons
45:   button_dismiss = new QPushButton( "Dismiss", this );
46:   connect( button_dismiss, SIGNAL( clicked() ),
47:       this, SLOT( slotDismiss() ) );
48:   button_cancel = new QPushButton( "Cancel", this );
49:   connect( button_cancel, SIGNAL( clicked() ),
50:       this, SLOT( slotCancel() ) );
51:
52:   // arrange the buttons
53:   QBoxLayout *buttons = new QHBoxLayout( topLayout, 2 );
54:   buttons->addWidget( button_dismiss );
55:   buttons->addWidget( button_cancel );
56:
57:   setMinimumWidth( 250 );
58: }
59:
60: void EditDialog::slotDismiss()
61: {
62:   name = edit_name->text();
63:   email = edit_email->text();
64:   phone = edit_phone->text();
65:   done ( 1 );
66: }
67:
68: void EditDialog::slotCancel()
69: {
70:   done ( 0 );
71: }
72:
73: AddDialog::AddDialog( QWidget* parent ):
74:   EditDialog( parent, QString::null, QString::null, QString::null )
75: {
```

```
76:   setCaption( "Add" );
77: }
```

Listing 3. edit.cpp

Widget-widget untuk EditDialog diatur dengan menggunakan layout yang ditata vertikal, jadi memanfaatkan QVBoxLayout. Namun demikian, untuk menghasilkan penataan kotak isian nama, e-mail, dan nomor telefon, maka disusunlah layout lain (kali ini dengan QGridLayout) di dalam baris pertama layout yang utama (disebut sebagai `topLayout` pada program). Hal yang demikian sering dinamakan sebagai layout bertingkat. Bila Anda cukup teliti, layout bertingkat juga digunakan sekali lagi untuk mengatur peletakan tombol Dismiss dan Cancel.

Terlihat bahwa baik `slotDismiss` maupun `slotCancel` memanggil fungsi `QDialog::done()`. Argumen yang diberikan ke fungsi ini akan menjadi nilai kembali (*return value*) dari kotak dialog tersebut. Lumrahnya, nilai ini dipergunakan kelak di program utama untuk memeriksa apakah user menekan Dismiss atau membatalkan semuanya karena menekan Cancel.

Pada konstruktor EditDialog terdapat tiga buah string yang dikirimkan yang akan berfungsi sebagai nilai awal untuk kotak isian, masing-masing untuk nama, e-mail, dan nomor telfon. EditDialog juga mempunyai variabel anggota (yang bersifat public) untuk menampung hasil penyuntingan dari user.

Bagaimana untuk AddDialog ? Sangat sederhana, AddDialog hanyalah EditDialog dengan nilai isian awal untuk nama, e-mail, dan nomor telfon sama dengan string null. Yang perlu diubah hanya title dari dialog ini (bukan lagi *Edit*, tetapi menjadi *Add*).

Begitu kotak dialog untuk mengedit atau menambah data sudah tersedia, kini saatnya memodifikasi program utama `kawan.cpp` agar dapat menggunakan kotak dialog tersebut.

```
 1: // kawan.cpp - Simple contact manager
 2:
 3: #include <qapplication.h>
 4: #include <qfile.h>
 5: #include <qlistview.h>
 6: #include <qmenubar.h>
 7: #include <qmessagebox.h>
 8: #include <qstatusbar.h>
 9: #include <qtextstream.h>
10:
11: #include "edit.h"
12:
13: #include "kawan.moc"
14:
15: const char* datafile = "kawan.dat";
16:
17: Kawan::Kawan() : QMainWindow( 0, "Kawan", WDestructiveClose )
18: {
19:   entries.setAutoDelete( TRUE );
20:
21:   initMenu();
```

```
 22:
 23:   list = new QListView( this );
 24:   list->addColumn( "Name" );
 25:   list->addColumn( "E-mail" );
 26:   list->addColumn( "Phone" );
 27:   list->setAllColumnsShowFocus( TRUE );
 28:   setCentralWidget( list );
 29:
 30:   loadData();
 31:
 32:   resize( 400, 250 );
 33: }
 34:
 35: void Kawan::loadData()
 36: {
 37:   QFile f( datafile );
 38:   if ( !f.open( IO_ReadOnly ) )
 39:     return;
 40:
 41:   entries.clear();
 42:
 43:   QTextStream stream( &f );
 44:   while ( !stream.eof() )
 45:     {
 46:       QStringList s = QStringList::split( ":", stream.readLine(),
TRUE );
 47:       QString name = s[0].stripWhiteSpace();
 48:       QString email = s[1].stripWhiteSpace();
 49:       QString phone = s[2].stripWhiteSpace();
 50:       if( !name.isEmpty() )
 51:    {
 52:      QListViewItem* e = new QListViewItem( list, name, email, phone
);
 53:      entries.append( e );
 54:    }
 55:     }
 56:
 57:   f.close();
 58:
 59:   statusBar()->message( QString("%1 entries").arg( entries.count()
), 2000 );
 60: }
 61:
 62: void Kawan::saveData()
 63: {
 64:   QFile f( datafile );
 65:   if ( !f.open( IO_WriteOnly ) )
 66:     return;
 67:
 68:   QTextStream stream( &f );
 69:
 70:   QListIterator<QListViewItem> it( entries );
 71:   for(; it.current(); ++it )
 72:     {
 73:       QListViewItem* e = it.current();
 74:       QString s = QString( "%1:%2:%3" ).arg( e->text(0) ).
 75:    arg( e->text(1) ).arg( e->text(2) );
 76:       if( !e->text(0).isEmpty() )
 77:    stream << s << endl;
```

```
 78:     }
 79:
 80:   f.close();
 81: }
 82:
 83: void Kawan::initMenu()
 84: {
 85:     QPopupMenu * entry = new QPopupMenu( this );
 86:     menuBar()->insertItem( "&Entry", entry );
 87:     entry->insertItem( "Add", this, SLOT( slotEntryAdd() ) );
 88:     entry->insertItem( "Delete", this, SLOT( slotEntryDelete() ) );
 89:     entry->insertItem( "Edit", this, SLOT( slotEntryEdit() ) );
 90:     entry->insertSeparator();
 91:     entry->insertItem( "Close", this, SLOT( close() ) );
 92:
 93:     QPopupMenu * help = new QPopupMenu( this );
 94:     menuBar()->insertSeparator();
 95:     menuBar()->insertItem( "&Help", help );
 96:     help->insertItem( "&About...", this, SLOT( slotAbout())  );
 97: }
 98:
 99: void Kawan::slotEntryAdd()
100: {
101:     AddDialog *d = new AddDialog( this );
102:     if( d->exec() )
103:       {
104:    QListViewItem* e = new QListViewItem( list, d->name,
105:                                          d->email, d->phone );
106:    entries.append( e );
107:    saveData();
108:    statusBar()->message( "Added", 1000 );
109:       }
110:     delete d;
111: }
112:
113: void Kawan::slotEntryEdit()
114: {
115:   QListViewItem* e = list->currentItem();
116:
117:   if( e )
118:     {
119:       EditDialog *d = new EditDialog( this, e->text(0),
120:                                  e->text(1), e->text(2) );
121:       if ( d->exec() )
122:    {
123:      e->setText( 0, d->name );
124:      e->setText( 1, d->email );
125:      e->setText( 2, d->phone );
126:      saveData();
127:      statusBar()->message( "Modified", 1000 );
128:    }
129:       delete d;
130:     }
131: }
132:
133: void Kawan::slotEntryDelete()
134: {
135:   QListViewItem* e = list->currentItem();
136:   if( e )
```

```
137:     {
138:       list->takeItem( e );
139:       saveData();
140:       statusBar()->message( "Deleted", 1000 );
141:     }
142: }
143:
144: void Kawan::slotAbout()
145: {
146:   QMessageBox::about( this, "Kawan",
147:                 "<b>Kawan 0.2</b>"
148:                 "<p>Simple contact manager</p>"
149:                 "<p>Programmed by Ariya Hidayat<br>"
150:                 "ariya@infolinux.co.id</p>" );
151: }
```

Listing 4. kawan.cpp

Dari listing `kawan.cpp` di atas, terdapat perubahan berarti dari `slotEntryAdd` dan `slotEntryEdit`. Keduanya tidak lagi kosong tetapi sudah berisi perintah-perintah. Prinsipnya sederhana: buat dialognya (entah dengan EditDialog atau AddDialog) dan panggil fungsi `QDialog:exec()`. Karena fungsi ini akan menghasilkan nilai kembalian tertentu, sementara pada implementasi EditDialog dan AddDialog nilai kembalian ini diatur sama dengan non-zero bila tombol Dismiss diaktifkan, maka dapat diketahui dengan mudah apakah Dismiss atau Cancel yang dipilih user. Selanjutnya, jika memang bukan Cancel, informasi field nama, e-mail, dan nomor telfon akan diambil dari kotak dialog tersebut. Untuk `slotEntryEdit` field-field yang baru akan menggantikan isi field yang lama. Untuk `slotEntryAdd`, field-field ini akan dijadikan satu entri data yang baru. Tidak terlalu susah, bukan ?

Patut juga dicatat, fungsi `saveData` segera dipanggil setelah terjadi perubahan data dengan tujuan agar data-data yang baru langsung tersimpan ke file `kawan.dat`.

Bila dibandingkan dengan versi 0.1, pada fungsi `saveData` dan `loadData` ada perombakan kecil, yaitu pengabaian entri data yang isinya hanya *null-string*. Catatan: sebetulnya ini adalah bug kecil yang baru ditemukan penulis segera setelah tulisan naik cetak.

Program utama `main.cpp` berikut ini juga adalah file yang tidak mengalami perubahan apa-apa dibandingkan versi 0.1-nya.

```
 1: // main.cpp
 2:
 3: #include <qapplication.h>
 4: #include "kawan.h"
 5:
 6: int main( int argc, char ** argv )
 7: {
 8:   QApplication a( argc, argv );
 9:
10:   Kawan *w = new Kawan();
11:   w->show();
12:
```

```
 13:   a.connect( &a, SIGNAL(lastWindowClosed()), &a, SLOT(quit()) );
 14:
 15:   return a.exec();
 16: }
 17:
```

Listing 5. main.cpp

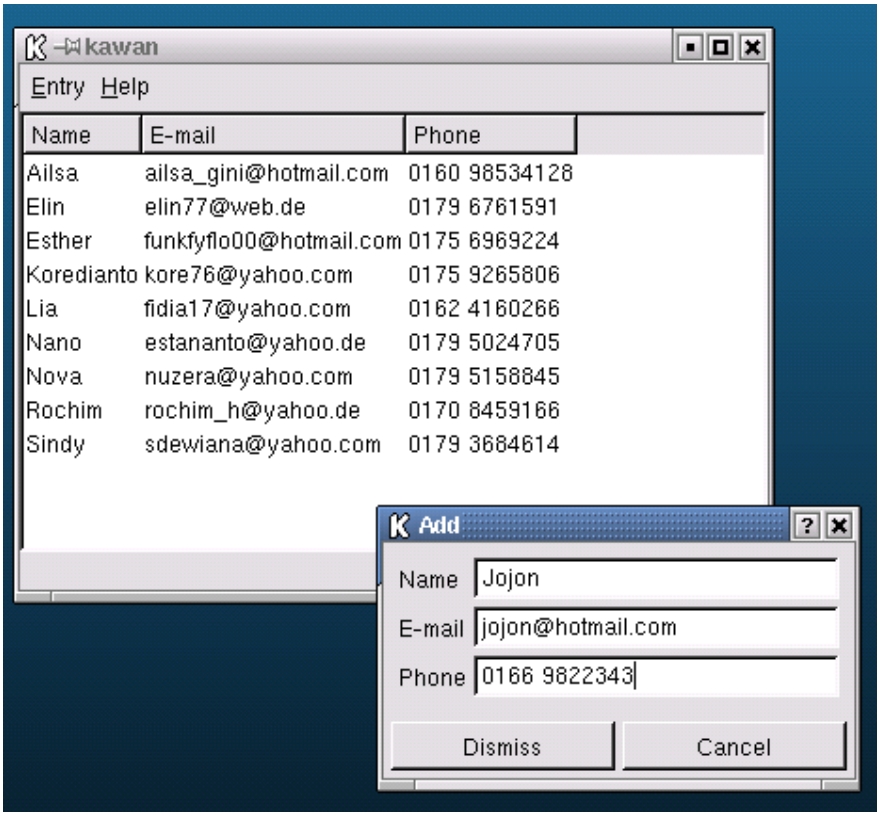
Untuk melakukan kompilasi program, digunakan Makefile berikut ini. Lagi-lagi tidak ada perubahan berarti pada Makefile ini kecuali penambahan `edit.moc` dan `edit.o`.

```
  1: QTDIR     = /usr/lib/qt2
  2: QTLIB     = $(QTDIR)/lib
  3: QTINCLUDE = $(QTDIR)/include
  4: QTMOC     = $(QTDIR)/bin/moc
  5:
  6: TARGET    = kawan
  7: OBJECTS   = kawan.o edit.o main.o
  8: MOCS      = kawan.moc edit.moc
  9:
 10: CC        = g++
 11: CFLAGS    = -Wall -O2
 12: LIBS      = -lqt
 13:
 14: $(TARGET): $(MOCS) $(OBJECTS)
 15:    $(CC) $(LFLAGS) -o $(TARGET) $(OBJECTS) $(LIBS) -L$(QTLIB)
-I$(QTINCLUDE)
 16:
 17: %.o: %.cpp
 18:    $(CC) $(CFLAGS) -I$(QTINCLUDE) -c $< -o $@
 19:
 20: %.moc: %.h
 21:    $(QTMOC) $< -o $@
 22:
 23: clean:
 24:    rm $(OBJECTS) $(TARGET) $(MOCS)
 25:
 26:
 27:
```

Listing 6. Makefile

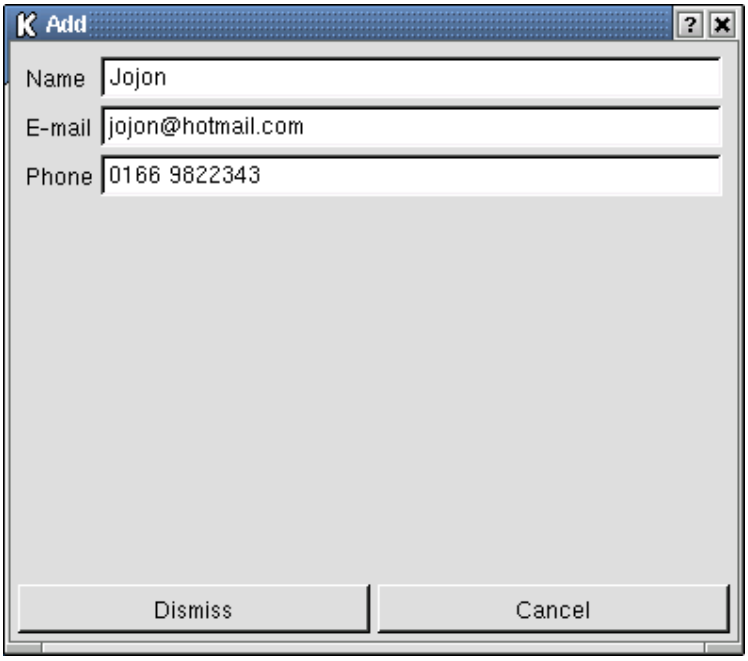
Jika Anda sudah sukses melakukan kompilasi dan menghasilkan file executable `kawan`, maka program baru ini siap untuk digunakan. Begitu Anda jalankan, tampilan awalnya tidak berbeda sama sekali dengan versi sebelumnya. Cobalah aktifkan *About* dari menu *Help*, kotak dialog About yang tampil akan menunjukkan bahwa progam Kawan ini sekarang sudah bernomor versi 0.2.

Kini pilihlah *Add* dari menu *Entry* sehingga tampil kotak dialog Add yang bisa diisi untuk memasukkan entri data yang baru (Gambar 6).



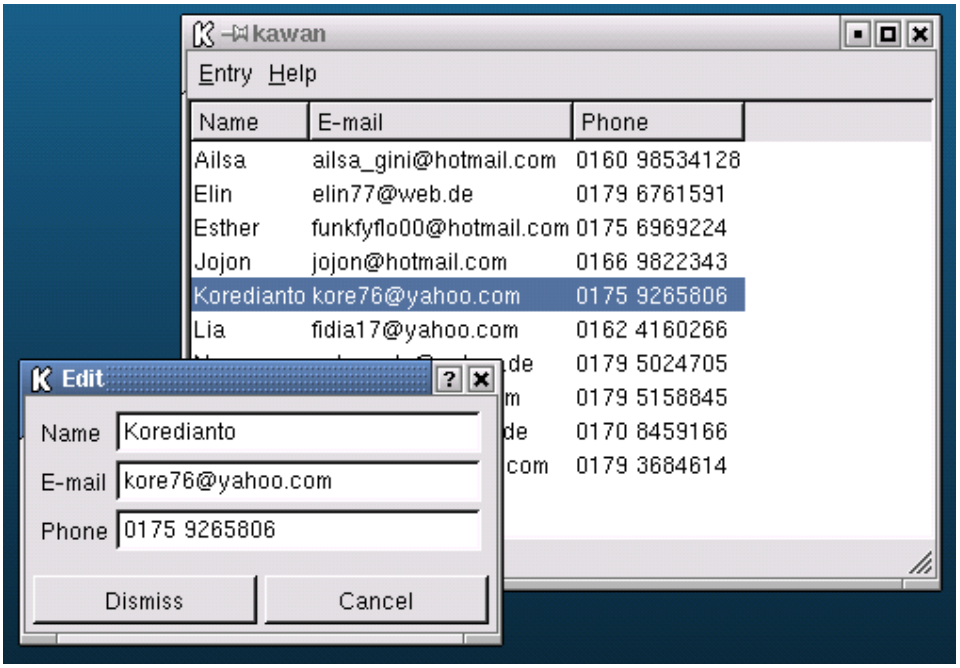
Gambar 6. Menambahkan Entri Baru

Kotak Dialog Add ini diatur dengan menggunakan QLayout, jadi Anda bisa perhatikan efeknya jika ukuran dialog ini diperbesar seperti ditunjukkan pada Gambar 7. Lihat bagaimana widget-widgetnya akan menyesuaikan diri sesuai dengan ruang yang tersedia.



Gambar 7. Perubahan Ukuran AddDialog

Mengedit data bisa dilakukan dengan menyorot data yang diinginkan dan memilih *Edit* dari menu *Entry*. Adapun tampilan dari kotak dialog Edit ini bisa dilihat berikut ini. Begitu field-fieldnya sudah dimodifikasi dan *Dismiss* diaktifkan, maka bisa dilihat bahwa daftar yang ada sudah menyajikan perubahan yang baru saja dibuat oleh user.



Gambar 8. Mengedit Data

Nah, kini fungsi program Kawan sebagai *contact manager* sudah lengkap. Sebagai latihan, Anda bisa memodifikasi program ini agar tidak hanya menyimpan nama, e-mail, dan nomor telfon, tetapi juga misalnya alamat, pekerjaan, tanggal lahir, dan sebagainya.

Latihan: Melayout Gallery

Masih ingatkan Anda akan program Gallery yang pernah disajikan sebagai contoh di seri pertama tulisan ini ? Bila Anda perhatikan dengan teliti, widget-widget program Gallery ini sama sekali tidak dilayout, melainkan hanya diletakkan begitu saja, sudah ditentukan di posisi pixel mana dia akan berada.

Setelah membahas tuntas mengenai aneka ragam penggunaan QLayout, sebagai latihan maka Anda bisa merombak program Gallery tersebut supaya kini tertata dengan baik (atau bisa dikatakan sudah merujuk ke konsep *geometry-management*).

Sekedar petunjuk, widget-widget ini bisa disusun secara vertikal sebagai 6 baris. Khusus untuk baris pertama, kedua, dan terakhir, maka bisa dilakukan layout bertingkat untuk meletakkan label, edit box, dan button dengan baik.

Bonus: Dari Source Code ke Kode HTML

Pernah ada yang bertanya, bagaimana saya menyiapkan naskah untuk tulisan ini ? Nah, sebagaimana kebiasaan saya pribadi, tulisan ini disusun dalam format HTML dengan menggunakan Emacs sebagai penyunting teksnya. Tidak ada yang aneh dalam hal ini kecuali mungkin sedikit hal unik mengenai penyisipan source code dari file-file yang dibahas. Untuk alasan keterbacaan, nomor baris harus dibubuhkan pada tiap-tiap baris dalam file source code, ini membuat file tersebut tidak bisa disisipkan begitu saja ke Emacs. Trik yang elegan adalah membuat kode-kode Lisp untuk melakukan hal ini secara otomatik. Akan tetapi, karena tulisan ini bercerita tentang pemrograman Qt/KDE, tidak ada salahnya menyusun program kecil untuk kebutuhan ini yang kemudian dinamakan sebagai *source2html*.

```
 1: // source2html.cpp
 2:
 3: #include <iostream.h>
 4: #include <qfile.h>
 5: #include <qregexp.h>
 6: #include <qstring.h>
 7: #include <qtextstream.h>
 8:
 9: int main( int argc, char **argv )
10: {
11:     if( argc < 2 ) return -1;
12:
13:     QString filename = argv[1];
14:
15:     QFile f( filename );
16:     if ( !f.open( IO_ReadOnly ) )
17:         return -1;
18:
19:     cout << "<html><head><title>" << filename << "</title></head>"
<< endl;
```

```
20:     cout << "<body>" << endl;
21:     cout << "<pre>"<< endl;
22:
23:     QTextStream t(&f);
24:     int i = 1;
25:     while ( !t.eof() )
26:     {
27:         QString s = t.readLine();
28:         s.replace( QRegExp("&"), "&amp;" );
29:         s.replace( QRegExp("<"), "&lt;" );
30:         s.replace( QRegExp(">"), "&gt;" );
31:         cout << QString("%1: %2").arg( i, 3 ).arg( s ) << endl;
32:         i++;
33:     }
34:     f.close();
35:
36:     cout << "</pre>"<< endl;
37:     cout << "</body>"<< endl;
38:     cout << "</html>"<< endl;
39:
40:     return 0;
41: }
```

Listing 7. source2html.cpp

Tidak ada yang rumit dari program di atas. Prinsip kerjanya sederhana: buka file input, baca baris demi baris, cetak kembali dengan nomor baris. Mudah bukan ? Oh ya, baris 28-30 diperlukan karena karakter khusus seperti &, <, dan > harus dikodekan sebagai &, <, dan &gt.

Karena program ini kecil dan hanya satu file saja, kompilasi bisa dilakukan semudah ini:

```
g++ -o source2html source2html.cpp -lqt -I/usr/lib/qt2/include
-L/usr/lib/qt2/lib
```

Jangan lupa sesuaikan `/usr/lib/qt2` dengan direktori instalasi Qt pada sistem Anda.

Setelah terbentuk file executable `source2html`, maka utility kecil ini siap digunakan. Untuk menghasilkan versi HTML dari source code bernama `contoh.cpp`, gunakan perintah berikut:

```
source2html contoh.cpp
```

Kode-kode HTML akan tertampilkan ke layar (standard output). Anda bisa melakukan redirecting ke file ataupun diproses lagi dengan filter. Selamat mencoba !

Tanya Jawab

Tanya: Mengapa harus ada *geometry management*? Bukankah kita bisa meletakkan widget-widget pada posisi (per pixel) yang diinginkan ?
Jawab: Meski terlihat menyulitkan, *geometry management* bisa menjadi menguntungkan. Hal ini terjadi manakala widget-widget diatur dalam sebuah window yang bisa berubah-ubah ukurannya. Dengan meletakkan widget di posisi yang pasti,

perubahan ukuran window tidak akan berpengaruh apa-apa terhadap keadaan widget tersebut. Sebaliknya, menggunakan konsep *geometry management*, perubahan ukuran window bisa menyebabkan widget-widget tertentu beralih tempat, memanjang, atau memendek. Selain itu, perhatikan bahwa kebanyakan aplikasi KDE mendukung penggunaan banyak bahasa, tidak hanya Inggris. Sekarang bayangkan bila ukuran sebuah *push-button* hanya sebesar yang diperlukan captionnya - misalnya "Save", apa yang terjadi bila program ini kemudian mendukung bahasa lain, katakanlah bahasa Jerman atau bahasa Indonesia ? Caption yang sudah berubah - menjadi "Speichern" untuk bahasa Jerman bisa tidak muat pada push-button tersebut. Dengan *geometry-management*, hal ini tidak akan terjadi karena push-button ini akan menyesuaikan ukuran dirinya sendiri. Malahan, mungkin saja window yang menjadi ditumpangi button tersebut juga membesar dengan sendirinya.

Tanya: Saya terus gagal untuk melakukan compile.
Jawab: Salah satu kesalahan paling yang cukup sering dilakukan adalah meng-include file header dan bukan file moc. Periksa source code Anda, apakah meng-include-kan `edit.h` yang seharusnya adalah `edit.moc`.

Tanya: Saya memodifikasi program Kawan yang sudah saya buat sebelumnya, tetapi mengapa kerap kali muncul pesan kesalahan "Segmentation Fault". Bagaimana bisa demikian ?
Jawab: Dalam kebanyakan kasus, ini disebabkan ketidaksinkronan antara file hasil Meta Object Compiler (berekstensi .moc) dengan header file (bereksensi .h). Pemecahan: melakukan kompilasi ulang dengan menjalankan `make clean` sebelum `make`.

Tanya: Apakah QLayout bisa digunakan di window utama ?
Jawab: Tentu saja. Dalam contoh yang diketengahkan di bagian ini (dan juga bagian-bagian sebelumnya dari tulisan ini), window utama program masih begitu sederhananya dan hanya terdiri atas satu widget utama sehingga tidak perlu susah-susah dilayout. Akan tetapi, Anda tentu masih ingat program Gallery yang muncul di bagian pertama. Program Gallery ini menggunakan penataan manual untuk meletakkan widget, yaitu di posisi yang ditentukan per piksel. Dengan memanfaatkan QLayout, program Gallery bisa dimodifikasi agar menggunakan juga manajemen geometri yang sudah dikupas di atas.

Tanya: Bisakah layout digunakan bertingkat ?
Jawab: Sangat bisa. Jikalau Anda cermati, kotak dialog Edit dari program Kawan menggunakan layout bertingkat, yaitu layout di dalam layout. Untuk mendesain susunan widget yang kompleks, layout bertingkat bisa menjadi amat berguna. Namun, perhatikan bahwa layout bertingkat meningkatkan kerumitan program sehingga harus dipergunakan secara berhati-hati.

Tanya: Bagaimana menyisipkan ruang kosong fleksibel di antara widget ?
Jawab: Triknya adalah menggunakan widget yang bebas untuk berubah ukurannya, sesuai dengan ruang yang tersedia. Dengan demikian, widget inilah yang akan menjadi "ruang kosong;". Untuk ini, bisa dimanfaatkan widget dari kelas QWidget yang memang tidak ada bentuk tampilannya alias kosong saja, seolah tidak ada apa-apa di situ. Catatan: trik ini sudah digunakan di `edit.cpp` yang dibahas di tulisan ini, lihat saja baris 38-42.

Tanya: Nah, sekarang bagaimana menyisipkan ruang kosong yang tidak fleksibel di antara widget ?
Jawab: Masih menggunakan trik yang sama: bentuk sebuah widget dari QWidget, kemudian atur dengan `setSizePolicy` untuk memaksa ukurannya tidak berubah (yakni `QSizePolicy::Fixed`). Jangan lupa untuk menentukan ukuran widget ini dengan `resize`.

Tanya: Tentang program kecil source2html, mengapa tidak menggunakan Perl atau Python saja ?
Jawab: Seperti sudah disebutkan, program ini sekaligus berfungsi mendemonstrasikan Qt, karenanya dibuat dengan Qt. Pada praktisnya, Anda bisa saja meramu sebuah script pendek dalam Perl, Python, atau bahkan shell script biasa untuk fungsi yang sama.

Tanya: Siapakah empat nama yang disajikan di contoh QGridLayout ?
Jawab: Empat nama ini - bersama juga Albus Dumbledore - semestinya tidak asing untuk anak kecil. Akan tetapi tidak bisa dipungkiri juga bahwa tidak semua Muggle mengetahui keberadaan mereka...

Tutorial Qt (bagian 5)

Berbekal pengetahuan tentang Qt yang selama ini telah dibahas, sekarang adalah saatnya untuk menampilkan sebuah studi kasus pengembangan aplikasi contoh yang cukup berguna dalam kehidupan nyata, yaitu sebuah editor multifile bernama...

Expad

Sebuah editor teks dikatakan multifile jika dapat menyunting beberapa file sekaligus. Lazimnya kemampuan seperti ini dimiliki oleh aplikasi-aplikasi office, misalnya StarOffice atau Microsoft Office (di Windows). Yang akan diulas di sini tentu saja aplikasi yang lebih sederhana karena hanya bertugas mengedit file teks biasa saja, namun punya keunggulan menampung sejumlah file pada satu window aplikasi.

Mula-mula marilah kita definisikan spesifikasi dari editor yang akan dinamakan *Expad* ini. Syarat utama tentu bahwa editor dapat menampilkan dan menangani file-file teks sekaligus, dengan jumlah yang tidak dibatasi. Untuk mudahnya, sekian banyak file teks ini tidak dipisah-pisah dengan menggunakan window editor sendiri-sendiri. Akan lebih indah jikalau digunakan *tab* untuk memilih file yang akan diaktifkan. Kira-kira hal demikian serupa dengan memilih lembar kerja di aplikasi spreadsheet (seperti Microsoft Excel di Windows). Guna kemudahan user, tab ini harus diletakkan di bagian atas.

Untuk memilih-milih file yang akan diedit, Expad diharapkan dapat menampilkan struktur direktori pada sebuah panel tersendiri. Selain mendaftar direktori, panel ini juga menyajikan daftar file-file pada direktori yang aktif. Dengan panel ini, user bisa berpindah direktori dengan mudah dan juga dapat memilih file yang ingin disunting. Jadi begitu menyorot file tertentu dan mengkliknya, maka file ini akan dimuatkan ke editornya dan siap dimodifikasi.

Sebagai pelengkap, Expad juga akan memiliki toolbar untuk mengakses fungsi-fungsi yang sering diperlukan dengan mudah dan cepat. Adapun yang akan dimasukkan ke toolbar adalah fungsi operasi file (new, open, save) dan fungsi penyuntingan (cut, copy, paste, undo).

Dari deskripsi sederhana tentang fungsionalitas Expad ini mestinya sudah dapat disketsakan secara kasar apa-apa saja yang perlu diimplementasikan. Yang jelas dibutuhkan adalah window utama aplikasi harus selalu ada dan ini diturunkan dari widget QMainWindow. Untuk Expad, window utama ini akan disebut sebagai ExpadWindow. Dua elemen lain yang akan menjadi bawahan dari ExpadWindow adalah editor teks dengan tab dan panel yang menyajikan daftar direktori dan file. Yang pertama akan berupa widget ExpadTab dan yang kedua adalah DirList. Widget ExpadTab berasal dari widget QTabWidget yang sudah disediakan Qt yang memang khusus dipergunakan kalau ingin memanfaatkan tab-tab, lumrahnya dijumpai pada pembuatan kotak dialog yang kompleks. Sementara itu DirList mewarisi widget QListView, dimodifikasi untuk bisa mendaftar direktori/file. Tentang QListView, contoh pemakaiannya sudah pernah disinggung di bagian empat seri tulisan ini.

File header `expad.h` di bawah ini agaknya bisa berbicara lebih banyak daripada segerobak kata-kata.

```
 1: // expad.h
 2: #ifndef __EXPAD_H
 3: #define __EXPAD_H
 4:
 5: #include <qmainwindow.h>
 6: #include <qaction.h>
 7: #include <qwidget.h>
 8: #include <qlist.h>
 9: #include <qstring.h>
10: #include <qmultilineedit.h>
11: #include <qtabwidget.h>
12: #include <qprinter.h>
13: #include <qsplitter.h>
14: #include <qlistview.h>
15:
16: class Editor: public QMultiLineEdit
17: {
18:     Q_OBJECT
19:
20:   public:
21:     Editor( QWidget * parent=0, const char * name=0 );
22:     void load();
23:     void save();
24:     QString filename();
25:     QString title();
26:     void setFilename( const QString& fn );
27:
28:   private:
29:     QString m_filename;
30:     QString m_shortname;
31: };
32:
33: class DirListItem: public QListViewItem
34: {
35:   protected:
36:     bool m_isDir;
37:     bool m_isReadable;
38:     QString m_name;
39:
40:   public:
41:     DirListItem( QListView*, const QString&, bool, bool );
42:     QString text( int ) const;
43:     bool isDir(){ return m_isDir; }
44:     bool isReadable(){ return m_isReadable; }
45:     QString shortName(){ return m_name; }
46: };
47:
48: class DirList: public QListView
49: {
50:     Q_OBJECT
51:
52:   public:
53:     DirList( QWidget* parent=0, const char* name=0 );
54:     void setDir( const QString& );
55:     QString dir(){ return m_dir; }
```

```
 56:
 57:   protected slots:
 58:     void slotDoubleClicked( QListViewItem* item );
 59:
 60:   signals:
 61:     void selected( const QString& filename );
 62:
 63:   protected:
 64:     QString m_dir;
 65:
 66: };
 67:
 68: class ExpadTab: public QTabWidget
 69: {
 70:     Q_OBJECT
 71:
 72:    public:
 73:     ExpadTab( QWidget * parent=0, const char * name=0, WFlags f=0 ):
 74:       QTabWidget( parent, name, f ) {};
 75:
 76:     int count();
 77:
 78:     void addEditor( Editor* e, const QString& t );
 79:     Editor* currentEditor();
 80:     void removeEditor( Editor* e );
 81:     void open( const QString& filename = "" );
 82:     void save();
 83: };
 84:
 85: class ExpadWindow: public QMainWindow
 86: {
 87:     Q_OBJECT
 88:
 89: public:
 90:     ExpadWindow();
 91:
 92: protected:
 93:
 94:     void initActions();
 95:     void initMenu();
 96:     void initToolbar();
 97:
 98: private slots:
 99:
100:     void fileNew();
101:     void fileOpen();
102:     void fileSave();
103:     void fileSaveAs();
104:     void fileClose();
105:     void fileCloseAll();
106:     void fileQuit();
107:
108:     void editUndo();
109:     void editRedo();
110:     void editCut();
111:     void editCopy();
112:     void editPaste();
113:     void editSelectAll();
114:
```

```
115:     void about();
116:
117:     void tabChanged( QWidget* );
118:     void openFile( const QString& filename );
119:
120: private:
121:
122:     // main widgets
123:     QSplitter* splitter;
124:     ExpadTab *tabs;
125:     DirList* dirlist;
126:
127:     // actions
128:     QAction* ma_FileNew;
129:     QAction* ma_FileOpen;
130:     QAction* ma_FileSave;
131:     QAction* ma_FileSaveAs;
132:     QAction* ma_FileClose;
133:     QAction* ma_FileCloseAll;
134:     QAction* ma_FileQuit;
135:
136:     QAction* ma_EditUndo;
137:     QAction* ma_EditRedo;
138:     QAction* ma_EditCut;
139:     QAction* ma_EditCopy;
140:     QAction* ma_EditPaste;
141:     QAction* ma_EditSelectAll;
142:
143:     QAction* ma_About;
144: };
145:
146: #endif
```

Listing 1. expad.h

Bisa dilihat, ada beberapa widget yang harus dikonstruksi. Kelas Editor adalah kelas yang akan digunakan oleh ExpadTab untuk menangani satu file yang disunting. Karena Expad bisa membuka beberapa file sekaligus, akan ada satu instance dari Editor untuk masing-masing file. Dengan menggunakan antarmuka tab yang dimiliki ExpadTab (diwarisi dari QTabWidget), maka satu Editor akan berupa satu tab dari ExpadTab. Hal ini akan menghasilkan tampilan yang sesuai dengan rancangan.

Sementara itu widget DirList tidak seberapa kompleks, tidak lain karena DirList adalah QListView yang diperluas supaya punya kemampuan mendaftar direktori/file. Untuk kelengkapan widget ini, perlu adanya kelas DirListItem (diturunkan dari QListViewItem). Masing-masing item pada DirList, entah direktori ataupun file, akan merupakan instance dari DirListItem.

Lepas dari definisi kelas-kelas tadi, ada sejumlah `QAction` di ExpadWindow. Apakah sebenarnya QAction ini ? Sebelumnya, simak terlebih dahulu implementasi widget-widget yang digunakan Expad dalam file program utama `expad.cpp` berikut ini:

```
1: // expad.cpp - Multiple-file text editor - Ariya Hidayat 2001
2:
```

```
 3: #include <qaccel.h>
 4: #include <qaction.h>
 5: #include <qapplication.h>
 6: #include <qdir.h>
 7: #include <qfiledialog.h>
 8: #include <qfileinfo.h>
 9: #include <qkeycode.h>
10: #include <qlayout.h>
11: #include <qlistview.h>
12: #include <qmenubar.h>
13: #include <qmessagebox.h>
14: #include <qmultilineedit.h>
15: #include <qpopupmenu.h>
16: #include <qsplitter.h>
17: #include <qstatusbar.h>
18: #include <qtabwidget.h>
19: #include <qtabbar.h>
20: #include <qtextstream.h>
21: #include <qtoolbar.h>
22:
23: #include "expad.moc"
24: #include "icons.h"
25:
26: const char * icon_expad_xpm[] = {
27: "32 32 20 1",
28: "  c None",
29: ". c #808000",
30: "+ c #525254",
31: "@ c #000000",
32: "# c #FFFFFF",
33: "$ c #DCDCDC",
34: "% c #FFDCA8",
35: "& c #C3C3C3",
36: "* c #585858",
37: "= c #303030",
38: "- c #A0A0A0",
39: "; c #000000",
40: "> c #808080",
41: ", c #FFC0C0",
42: "' c #404000",
43: ") c #C0C000",
44: "! c #FFFFC0",
45: "~ c #FFA858",
46: "{ c #C0C0FF",
47: "] c #400000",
48: "        ************=@          ",
49: "        *&&&&&&&&&---'@          ",
50: "   ******&#########$&*>@         ",
51: "   *&&&&*&##########$*&>@        ",
52: "   *&###*&###########*#&>@       ",
53: "   *&###*&##---######*##$>@      ",
54: "   *&###*&###########*$##&>@     ",
55: "   *&###*&##------###*&$##$>@    ",
56: "   *&###*&###########>******=@   ",
57: "   *&###*&##------####$$&&->*@   ",
58: "   *&###*&#############$$&&->@   ",
59: "   *&###*&##############$$&&>@   ",
60: "   *&###*&##---------####$$&-@   ",
61: "   *&###*&################$$-@   ",
```

```
 62: "   *&###*&##-------------#$$&@  ",
 63: "   *&###*&###################-@  ",
 64: "   *&###*&##-------------###&@  ",
 65: "   *&###*&###################-@  ",
 66: "   *&###*&##----------######&@  ",
 67: "   *&###*&###################-@  ",
 68: "   *&###*&###################-=  ",
 69: "   *&###*&##-------------###-=  ",
 70: "   *&###*&###################-=  ",
 71: "   *&###*&##-------------###-=  ",
 72: "   *&###*&###################-=  ",
 73: "   *&###*&###################-=  ",
 74: "   *&###*&&&-----------------=  ",
 75: "   *&###@==================]=@  ",
 76: "   *&######################-=    ",
 77: "   *&######################-=    ",
 78: "   *&&--------------------=    ",
 79: "   ========================    "};
 80:
 81: static const char* icon_folder_xpm[]={
 82: "16 16 17 1",
 83: "  c None",
 84: ". c #020202",
 85: "+ c #A1741D",
 86: "@ c #CC943A",
 87: "# c #8D6819",
 88: "$ c #C5C2BE",
 89: "% c #2B2B2B",
 90: "& c #E2DCD5",
 91: "* c #484848",
 92: "= c #ACA69C",
 93: "- c #737373",
 94: "; c #7A5A12",
 95: "> c #6A6A6A",
 96: ", c #EEEEEE",
 97: "' c #CEA975",
 98: ") c #563A06",
 99: "! c #AD6A0E",
100: "    .*          ",
101: "   .$,**%       ",
102: "  *#)-=$,*.*-.  ",
103: "  %&,'#)-=$,,,. ",
104: "  %'@'&,'+)-=$,*",
105: "  %@!@'''&,'#)-*",
106: "  %@++!!@@'$&,'*",
107: "  %+;###+!!@@'&*",
108: "  *,,=#;##++!!'*",
109: "  *,,,,&=#;##+@*",
110: "  *,,,&&&&$=#;#*",
111: "  %$&&&&&$$$$$>*",
112: "   .%>$$$$$$$=>*",
113: "      .%>=$$==>*",
114: "         .%>==>*",
115: "            .%% "};
116:
117: // -------- implementation of Editor ---------
118:
119: Editor::Editor( QWidget * parent, const char * name ):
120:       QMultiLineEdit( parent, name )
```

```
121: {
122:   m_filename = m_shortname = "";
123:   setFont( QFont( "adobe-courier" ) );
124: }
125:
126: QString Editor::filename()
127: {
128:   return m_filename;
129: }
130:
131: QString Editor::title()
132: {
133:   return m_shortname;
134: }
135:
136: void Editor::setFilename( const QString& fn )
137: {
138:   m_filename = fn;
139:   QFileInfo fi( m_filename );
140:   m_shortname = fi.fileName();
141:   if( m_shortname.isEmpty())
142:     m_shortname = "Untitled";
143: }
144:
145: void Editor::load()
146: {
147:   if( filename().isEmpty() ) return;
148:
149:   QFile f( filename() );
150:
151:   if( f.open( IO_ReadOnly ) )
152:     {
153:       setAutoUpdate( FALSE );
154:       clear();
155:       QTextStream t( &f );
156:       while ( !t.eof() ) {
157:    QString s = t.readLine();
158:    append( s );
159:       }
160:       f.close();
161:       setAutoUpdate( TRUE );
162:       repaint();
163:       setEdited( FALSE );
164:     }
165: }
166:
167: void Editor::save()
168: {
169:   if( filename().isEmpty() ) return;
170:
171:   QFile f( filename() );
172:
173:   if ( f.open( IO_WriteOnly ) )
174:     {
175:       QTextStream t( &f );
176:       t << text();
177:       f.close();
178:     }
179:
```

```
180:   setEdited( FALSE );
181: }
182:
183: // -------- implementation of DirListItem ---------
184:
185: DirListItem::DirListItem( QListView* parent, const QString& name,
186:                       bool dir, bool readable ):
187:   QListViewItem( parent, name )
188: {
189:   m_name = name;
190:   m_isDir = dir;
191:   m_isReadable = readable;
192:
193:   if( m_isDir ) setPixmap( 0, icon_folder_xpm );
194:
195: }
196:
197: QString DirListItem::text( int column ) const
198: {
199:   if( column == 0 ) return m_name;
200:   else if( column == 1 ) return m_isDir ? "Directory" : "File";
201:   else return QString::null;
202: }
203:
204: // -------- implementation of DirList ---------
205:
206: DirList::DirList( QWidget* parent, const char* name ):
207:   QListView( parent, name )
208: {
209:   m_dir = "";
210:   addColumn( "Name" );
211:   addColumn( "Type" );
212:
213:   setDir( "/" );
214:   setMinimumSize( 100, 50 );
215:   setSorting( -1 );
216:
217:   connect( this, SIGNAL( doubleClicked( QListViewItem* ) ),
218:       this, SLOT( slotDoubleClicked( QListViewItem* ) ) );
219: }
220:
221: void DirList::setDir( const QString& dir )
222: {
223:   QDir d( dir );
224:
225:   if( !d.exists() ) return;
226:
227:   m_dir = dir;
228:
229:   d.setSorting( QDir::Name | QDir::DirsFirst | QDir::IgnoreCase );
230:   const QFileInfoList *list = d.entryInfoList();
231:   QFileInfoListIterator it( *list );
232:   QFileInfo *fi;
233:
234:   clear();
235:   it.toLast();
236:
237:   while ( ( fi = it.current() ) )
238:     {
```

```
239:       if( fi->fileName() != "." )
240:    {
241:      QListViewItem* item;
242:      item  = new DirListItem( this, fi->fileName(),
243:                               fi->isDir(), fi->isReadable() );
244:    }
245:       --it;
246:     }
247: }
248:
249: void DirList::slotDoubleClicked( QListViewItem* i )
250: {
251:   DirListItem* item = (DirListItem*) i;
252:   if( item )
253:     if( item->isReadable() )
254:     {
255:       QString fullName = m_dir + "/" + item->shortName();
256:       if( item->isDir() ) setDir( fullName );
257:       else emit selected( fullName );
258:     }
259: }
260:
261: // -------- implementation of ExpadTab ---------
262:
263: int ExpadTab::count()
264: {
265:   return tabBar()->count();
266: }
267:
268: void ExpadTab::addEditor( Editor* e, const QString& t )
269: {
270:   if( !e ) return;
271:   addTab( e, t );
272:   setCurrentPage( count() - 1 );
273:   e->show();
274:   show();
275: }
276:
277: void ExpadTab::removeEditor( Editor* e )
278: {
279:   if( !e ) return;
280:   removePage( e );
281:   delete e;
282:   show();
283: }
284:
285: Editor* ExpadTab::currentEditor()
286: {
287:   return (Editor*) currentPage();
288: }
289:
290: // -------- implementation of ExpadWindow ---------
291:
292: const char* app = "Expad";
293:
294: ExpadWindow::ExpadWindow()
295:     : QMainWindow( 0, "ExpadWindow", WDestructiveClose )
296: {
297:   initActions();
```

```
298:   initMenu();
299:   initToolbar();
300:
301:   splitter = new QSplitter( this );
302:   setCentralWidget( splitter );
303:
304:   dirlist = new DirList( splitter);
305:   splitter->setResizeMode( dirlist, QSplitter::KeepSize );
306:   connect( dirlist, SIGNAL( selected( const QString& ) ),
307:       this, SLOT( openFile( const QString& ) ) );
308:
309:   tabs = new ExpadTab( splitter );
310:   connect( tabs, SIGNAL( currentChanged( QWidget* ) ),
311:       this, SLOT( tabChanged( QWidget* ) ) );
312:
313:   setIcon( icon_expad_xpm );
314:
315:   fileNew();
316:
317:   statusBar()->message( "Ready", 2000 );
318:   resize( 450, 400 );
319: }
320:
321: void ExpadWindow::initActions()
322: {
323:   ma_FileNew = new QAction( this );
324:   ma_FileNew->setText( "New" );
325:   ma_FileNew->setAccel( CTRL + Key_N );
326:   ma_FileNew->setIconSet( QPixmap( filenew_xpm ) );
327:   connect( ma_FileNew, SIGNAL( activated() ),
328:       this, SLOT( fileNew() ) );
329:
330:   ma_FileOpen = new QAction( this );
331:   ma_FileOpen->setText( "Open" );
332:   ma_FileOpen->setAccel( CTRL + Key_O );
333:   ma_FileOpen->setIconSet( QPixmap( fileopen_xpm ) );
334:   connect( ma_FileOpen, SIGNAL( activated() ),
335:       this, SLOT( fileOpen() ) );
336:
337:   ma_FileSave = new QAction( this );
338:   ma_FileSave->setText( "Save" );
339:   ma_FileSave->setAccel( CTRL + Key_S );
340:   ma_FileSave->setIconSet( QPixmap( filesave_xpm ) );
341:   connect( ma_FileSave, SIGNAL( activated() ),
342:       this, SLOT( fileSave() ) );
343:
344:   ma_FileSaveAs = new QAction( this );
345:   ma_FileSaveAs->setText( "Save As" );
346:   connect( ma_FileSaveAs, SIGNAL( activated() ),
347:       this, SLOT( fileSaveAs() ) );
348:
349:   ma_FileClose = new QAction( this );
350:   ma_FileClose->setText( "Close" );
351:   ma_FileClose->setAccel( CTRL + Key_W );
352:   connect( ma_FileClose, SIGNAL( activated() ),
353:       this, SLOT( fileClose() ) );
354:
355:   ma_FileCloseAll = new QAction( this );
356:   ma_FileCloseAll->setText( "Close All" );
```

```
357:   connect( ma_FileCloseAll, SIGNAL( activated() ),
358:       this, SLOT( fileCloseAll() ) );
359:
360:   ma_FileQuit = new QAction( this );
361:   ma_FileQuit->setText( "Quit" );
362:   ma_FileQuit->setAccel( CTRL + Key_X );
363:   connect( ma_FileQuit, SIGNAL( activated() ),
364:       this, SLOT( fileQuit() ) );
365:
366:   ma_EditUndo = new QAction( this );
367:   ma_EditUndo->setText( "Undo");
368:   ma_EditUndo->setAccel( CTRL + Key_Z );
369:   ma_EditUndo->setIconSet( QPixmap( editundo_xpm ) );
370:   connect( ma_EditUndo, SIGNAL( activated() ),
371:       this, SLOT( editUndo() ) );
372:
373:   ma_EditRedo = new QAction( this );
374:   ma_EditRedo->setText( "Redo");
375:   connect( ma_EditRedo, SIGNAL( activated() ),
376:       this, SLOT( editRedo() ) );
377:
378:   ma_EditCut = new QAction( this );
379:   ma_EditCut->setText( "Cut");
380:   ma_EditCut->setAccel( CTRL + Key_X );
381:   ma_EditCut->setIconSet( QPixmap( editcut_xpm ) );
382:   connect( ma_EditCut, SIGNAL( activated() ),
383:       this, SLOT( editCut() ) );
384:
385:   ma_EditCopy = new QAction( this );
386:   ma_EditCopy->setText( "Copy");
387:   ma_EditCopy->setAccel( CTRL + Key_C );
388:   ma_EditCopy->setIconSet( QPixmap( editcopy_xpm ) );
389:   connect( ma_EditCopy, SIGNAL( activated() ),
390:       this, SLOT( editCopy() ) );
391:
392:   ma_EditPaste = new QAction( this );
393:   ma_EditPaste->setText( "Paste");
394:   ma_EditPaste->setAccel( CTRL + Key_V );
395:   ma_EditPaste->setIconSet( QPixmap( editpaste_xpm ) );
396:   connect( ma_EditPaste, SIGNAL( activated() ),
397:       this, SLOT( editPaste() ) );
398:
399:   ma_EditSelectAll = new QAction( this );
400:   ma_EditSelectAll->setText( "Select All");
401:   ma_EditSelectAll->setAccel( CTRL + Key_A );
402:   connect( ma_EditSelectAll, SIGNAL( activated() ),
403:       this, SLOT( editSelectAll() ) );
404:
405:   ma_About = new QAction( this );
406:   ma_About->setText( "About...");
407:   connect( ma_About, SIGNAL( activated() ),
408:       this, SLOT( about() ) );
409: }
410:
411: // initialize menu
412: void ExpadWindow::initMenu()
413: {
414:   QPopupMenu *file_menu = new QPopupMenu( this );
415:   menuBar()->insertItem( "&File", file_menu );
```

```
416:   ma_FileNew->addTo( file_menu );
417:   ma_FileOpen->addTo( file_menu );
418:   file_menu->insertSeparator();
419:   ma_FileSave->addTo( file_menu );
420:   ma_FileSaveAs->addTo( file_menu );
421:   file_menu->insertSeparator();
422:   ma_FileClose->addTo( file_menu );
423:   ma_FileCloseAll->addTo( file_menu );
424:   file_menu->insertSeparator();
425:   ma_FileQuit->addTo( file_menu );
426:
427:   QPopupMenu *edit_menu = new QPopupMenu( this );
428:   menuBar()->insertSeparator();
429:   menuBar()->insertItem( "&Edit", edit_menu );
430:   ma_EditUndo->addTo( edit_menu );
431:   ma_EditRedo->addTo( edit_menu );
432:   edit_menu->insertSeparator();
433:   ma_EditCut->addTo( edit_menu );
434:   ma_EditCopy->addTo( edit_menu );
435:   ma_EditPaste->addTo( edit_menu );
436:   edit_menu->insertSeparator();
437:   ma_EditSelectAll->addTo( edit_menu );
438:
439:   QPopupMenu *help_menu = new QPopupMenu( this );
440:   menuBar()->insertSeparator();
441:   menuBar()->insertItem( "&Help", help_menu );
442:   ma_About->addTo( help_menu );
443: }
444:
445: // initialize toolbar
446: void ExpadWindow::initToolbar()
447: {
448:   QToolBar* toolbar = new QToolBar( this );
449:   ma_FileNew->addTo( toolbar );
450:   ma_FileOpen->addTo( toolbar );
451:   ma_FileSave->addTo( toolbar );
452:   toolbar->addSeparator();
453:   ma_EditCut->addTo( toolbar );
454:   ma_EditCopy->addTo( toolbar );
455:   ma_EditPaste->addTo( toolbar );
456:   ma_EditUndo->addTo( toolbar );
457:   QWidget* spacer = new QWidget( toolbar );
458:   spacer->setSizePolicy( QSizePolicy( QSizePolicy::Expanding,
459:                                       QSizePolicy::Expanding ) );
460:   setRightJustification( TRUE );
461: }
462:
463: // opens the given file as new document
464: void ExpadWindow::openFile( const QString& filename )
465: {
466:   if( !filename.isEmpty() )
467:     {
468:       Editor* editor = new Editor( tabs );
469:       editor->setFilename( filename );
470:       editor->load();
471:       tabs->addEditor( editor, editor->title() );
472:     }
473: }
474:
```

```
475: // creates a new blank document
476: void ExpadWindow::fileNew()
477: {
478:   Editor* editor = new Editor( tabs );
479:   editor->setFilename( QString::null );
480:   tabs->addEditor( editor, editor->title() );
481: }
482:
483: // opens and loads a document
484: void ExpadWindow::fileOpen()
485: {
486:   QString filename;
487:   filename  = QFileDialog::getOpenFileName( dirlist->dir(),
488:                                             QString::null, this);
489:   openFile( filename );
490: }
491:
492: // saves active document
493: void ExpadWindow::fileSave()
494: {
495:   Editor* editor = tabs->currentEditor();
496:   if( !editor) return;
497:
498:   if( editor->filename().isEmpty() ) fileSaveAs();
499:   editor->save();
500: }
501:
502: // saves document to a different name
503: void ExpadWindow::fileSaveAs()
504: {
505:   Editor* editor = tabs->currentEditor();
506:   if( !editor) return;
507:
508:   QString filename;
509:   filename  = QFileDialog::getSaveFileName( dirlist->dir(),
510:                                             QString::null, this);
511:   if( !filename.isEmpty() )
512:     {
513:       editor->setFilename( filename );
514:       editor->save();
515:       tabs->changeTab( editor, editor->title() );
516:     }
517: }
518:
519: // closes active document
520: void ExpadWindow::fileClose()
521: {
522:   Editor* editor = tabs->currentEditor();
523:   if( !editor ) return;
524:
525:   if( editor->edited() )
526:     {
527:       // confirm user
528:       QString msg = QString("The document <b>%1</b> has been changed
since "\
529:                             "the last save.").arg( editor->title() );
530:       int choice = QMessageBox::information( this, "Confirm", msg,
531:                                              "Save Now", "Discard",
"Cancel",
```

```
532:                                          0, 1 );
533:       if( choice == 2 ) return;
534:       if( choice == 0 ) fileSave();
535:     }
536:   tabs->removeEditor( editor );
537: }
538:
539: // closes all document
540: void ExpadWindow::fileCloseAll()
541: {
542:   while( tabs->currentEditor() )
543:     {
544:       Editor* editor = tabs->currentEditor();
545:
546:       if( editor->edited() )
547:    {
548:      // confirm user
549:      QString msg = QString("The document <b>%1</b> has been changed
since "\
550:                            "the last save.").arg( editor->title() );
551:      int choice = QMessageBox::information( this, "Confirm", msg,
552:                                             "Save Now", "Discard",
"Cancel",
553:                                             0, 1 );
554:      if( choice == 2 ) return;
555:      if( choice == 0 ) fileSave();
556:    }
557:       tabs->removeEditor( editor );
558:     }
559: }
560:
561: // exits program
562: void ExpadWindow::fileQuit()
563: {
564:   fileCloseAll();
565:   close();
566: }
567:
568: // reverts last action
569: void ExpadWindow::editUndo()
570: {
571:   Editor *e = tabs->currentEditor();
572:   if( e ) e->undo();
573: }
574:
575: // redoes last action
576: void ExpadWindow::editRedo()
577: {
578:   Editor *e = tabs->currentEditor();
579:   if( e ) e->redo();
580: }
581:
582: // cuts selected text
583: void ExpadWindow::editCut()
584: {
585:   Editor *e = tabs->currentEditor();
586:   if( e ) e->cut();
587: }
588:
```

```
589: // copies selected text
590: void ExpadWindow::editCopy()
591: {
592:   Editor *e = tabs->currentEditor();
593:   if( e ) e->copy();
594: }
595:
596: // pastes from clipboard
597: void ExpadWindow::editPaste()
598: {
599:   Editor *e = tabs->currentEditor();
600:   if( e ) e->paste();
601: }
602:
603: // selects all text
604: void ExpadWindow::editSelectAll()
605: {
606:   Editor *e = tabs->currentEditor();
607:   if( e ) e->selectAll();
608: }
609:
610: // displays program information
611: void ExpadWindow::about()
612: {
613:   QMessageBox::about( this, QString("About %1").arg( app ),
614:                 "<b>" + QString( app ) + "</b>" +
615:                 "<p>Multiple-file text editor</p>"
616:                 "<p>Programmed by Ariya Hidayat<br>"
617:                 "ariya@infolinux.co.id</p>" );
618: }
619:
620: void ExpadWindow::tabChanged( QWidget* )
621: {
622:   Editor* editor = tabs->currentEditor();
623:   if( !editor ) return;
624:   setCaption( editor->title() + " - " + app );
625: }
```

Listing 2. expad.cpp

Walaupun terbilang agak panjang, sesungguhnya file `expad.cpp` dapat dibagi atas beberapa bagian utama, masing-masing untuk implementasi kelas yang berbeda.

Kelas Editor mewakili widget utama tempat user mengedit isi dari file. Untuk mudahnya (sebagaimana bisa dicermati dari file header sebelumnya), Editor ini hanya merupakan modifikasi kecil dari widget QMultiLineEdit. Dibandingkan QMultiLineEdit, widget Editor ditambahkan fungsi untuk memudahkan menyimpan dan mengambil isi dari file teks yang sedang disunting. Terdapat juga fungsi `filename()` yang akan memberikan nama file yang bersesuaian dengan Editor tersebut. Bila ini adalah widget Editor yang baru dibuat dan isinya belum pernah disimpan ke file, `filename()` akan mengembalikan string kosong.

Penyunting teks lumrahnya bekerja dengan font yang sifatnya *fixed-width*, yakni yang lebar per karakternya sama semua, tidak bergantung dari karakter itu sendiri. Jadi baik huruf m maupun huruf i akan menyita ruang yang sama. Font semacam Courier tergolong

font yang seperti ini. Karenanya, dari awal widget Editor mengatur agar font yang digunakan adalah font *adobe-courier* yakni salah satu ragam font Courier yang biasanya sudah terinstal di sistem Linux. Umpamanya Anda ternyata tidak punya font ini, silakan ganti dengan yang tersedia, misalnya *bitstream-courier* atau malah *Courier New* jika Anda memasang font-font yang TrueType.

Kelas DirListItem akan merepresentasikan informasi mengenai satu item yang ditampilkan di widget DirList, baik untuk direktori maupun untuk file. Sebagai kelas pembantu untuk widget DirList, DirListItem akan memberikan keterangan dalam bentuk dua kolom: yang pertama untuk nama file atau direktori dan yang kedua untuk jenisnya, "Directory" ataukah "File". Guna membedakan antara file dan direktori, kelas DirListItem akan mengatur agar sebuah pixmap kecil bergambar folder dibubuhkan di bagian kiri jika item tersebut merupakan sebuah direktori.

Widget DirList merupakan panel yang mendaftar isi sebuah direktori tertentu. Hal ini sendiri dilakukan dengan menggunakan `QFileInfoList` sebagaimana bisa dilihat di fungsi `DirList::setDir()`. Fungsi `QDir::entryInfoList` sendiri akan mengembalikan item-item yang dimiliki oleh direktori yang dimaksud dan kemudian bisa ditelusuri menggunakan iterator bernama `QFileInfoListIterator`. Dari hasil penjelajahan iterasi ini akan dibentuk sejumlah DirListItem yang mewakili masing-masing item yang ditemukan (*quiz untuk pembaca: mengapa entri "." diabaikan?*). Perhatikan bahwa karena membuat item baru QListView (dan juga DirList) akan menyebabkan item tersebut *ditambahkan* di akhir daftar item yang ada sehingga item yang dibuat pertama akan tampil paling bawah. Karenanya, penelusuran iterator harus dilakukan dari akhir ke awal agar nantinya justru item yang harusnya urutannya awal tetap tampil paling atas.

Penting pula untuk diterangkan bagaimana DirList menangani klik-ganda dari user ketika menyoroti sebuah item. Bila item tersebut berupa file, maka DirList akan membangkitkan signal bernama `selected( const QString& )`, lihat lagi file header `expad.h`. Kelak signal ini akan dihubungkan dengan slot yang tepat sehingga menghasilkan aksi memuat file yang diklik ke editor. Sementara itu kejadiannya akan berbeda jika item yang diklik adalah direktori. DirList segera beralih ke direktori yang dimaksud, tentu apabila pengaturan permission mengijinkan user masuk ke dalamnya. Dalam kasus direktori ini, tidak perlu ada signal yang dipicu.

Melihat definisi widget ExpadTab di file header `expad.h` jelaslah bahwa widget ini persis seperti induknya, QTabWidget, dengan hanya fungsi-fungsi ekstra untuk menambahkan sebuah widget Editor baru menjadi salah satu tabnya atau menghapus tab yang sudah ada. Tab-tab di QTabWidget biasa disebut *page* dan inilah yang akan di-cast ke widget Editor. Sederhana saja, bukan ?

Fungsionalitas paling berat ada pada ExpadWindow sebagai widget yang merupakan window utama. ExpadWindow punya sejumlah slot yang berfungsi melakukan operasi file, penyuntingan, dan lain-lain.

Begitu banyak QAction yang ada di ExpadWindow mencerminkan pendekatannya yang sedikit berbeda soal penanganan input dari user. Sebuah QAction sendiri adalah abstraksi

dari suatu tindakan tertentu yang berasal dari user. Contohnya, `ma_FileNew` adalah action untuk membuat dokumen baru. Selain melalui menu *File*, *New*, user dapat juga memicu action ini dari tombol yang ada di toolbar maupun menggunakan keyboard dengan shortcut tertentu, dalam contoh ini adalah Ctrl+N. Keuntungannya adalah bahwa satu action saja bisa digunakan sebagai representasi item di menu program, button di toolbar, dan sebagai tombol shortcut sehingga terasa cukup mudah dan praktis.

Dalam fungsi `ExpadWindow::initMenu()` dan `ExpadWindow::initToolbar()`, sejumlah action yang diciptakan di `ExpadWindow::initActions()` disisipkan ke menu utama dan ke toolbar hanya dengan memanggil fungsi `addTo` dari masing-masing action. Anda tidak perlu lagi membuat item menu khusus atau tombol di toolbar, sebagaimana yang sudah-sudah (lihat lagi misalnya program QSimpleApp di bagian kedua seri tulisan ini). Selain menyederhanakan, teknik ini akan mengurangi duplikasi kode program yang tidak perlu.

Soal urusan layout widget, ExpadWindow tidak menggunakan cara-cara layout yang pernah dibahas sebelumnya. Ada dua elemen utama dalam satu window, yaitu panel direktori (DirList) dan editor (ExpadTab). Sudah sewajarnya kedua elemen ini tampil saling bersisian, namun dengan sebuah pembatas (*splitter*) yang dapat digerakkan untuk mengatur pembagian ruangan bagi keduanya. Dengan Qt, hal demikian menjadi mudah berkat widget QSplitter. Dua widget lain yang ingin dipisahkan tinggal menjadikan si splitter sebagai parent-nya.

Di konstruktor ExpadWindow dijumpai pemanggilan fungsi `setIcon()`. Sebenarnya ini adalah untuk mengatur icon dari window utama, terlihat ketika Expad sudah dieksekusi. Kelak icon yang sama juga muncul pada kotak dialog About.

Pada slot-slot yang mengurusi file, yakni `fileNew()`, `fileOpen()`, `fileSave()`, `fileSaveAs()`, `fileClose()`, serta `fileCloseAll()`, sebagian besar hanya bertindak sebagai jembatan untuk memanggil fungsi-fungsi yang tepat dari Editor ataupun ExpadTab. Yang agak berbeda barangkali hanya fungsi `fileClose()` dan `fileCloseAll()` yang harus mengkonfirmasi user terlebih dahulu tatkala ada dokumen yang belum disimpan namun hendak ditutup.

Slot-slot lainnya di ExpadWindow berkenaan dengan operasi penyuntingan. Yang ini malah jauh lebih sederhana, hanya memanggil fungsi penyuntingan yang sama dari salah satu editor. Ikhwal mana editor yang aktif dideteksi dengan `ExpadTab::currentEditor()`.

Jangan terlewatkan juga bahwa untuk icon-icon di toolbar, `expad.cpp` ini membutuhkan file header `icons.h`. File ini tidak disertakan lagi di sini (untuk alasan penghematan tempat) karena Anda bisa mengambil file `icons.h` yang sama seperti yang sudah dicantumkan di bagian kedua seri tulisan ini.

Kalaulah Anda masih kebingungan dengan beberapa kelas Qt yang dipergunakan dalam Expad, dokumentasi referensi Qt memberikan keterangan panjang lebar mengenai semuanya. Jangan segan-segan untuk senantiasa melongoknya !

Adapun program utamanya yang cukup beberapa belas baris kode adalah:

```
 1: // Expad
 2:
 3: #include <qapplication.h>
 4: #include "expad.h"
 5:
 6: int main( int argc, char ** argv )
 7: {
 8:   QApplication a( argc, argv );
 9:
10:   QMainWindow * w = new ExpadWindow();
11:   w->show();
12:
13:   a.connect( &a, SIGNAL(lastWindowClosed()), &a, SLOT(quit()) );
14:
15:   return a.exec();
16: }
```

Listing 3. main.cpp

Sebagaimana yang sudah-sudah, fungsi `main.cpp` di atas tidak kurang tidak lebih adalah menjalinkan window utama ExpadWindow ke dalam sebuah aplikasi utuh.

Berikutnya, inilah Makefile yang digunakan untuk memudahkan proses *compile-and-link*. Mudah-mudahan Anda tidak jenuh karena Makefile ini pada pokoknya selalu saja sama dari dulu, yang disesuaikan hanya bagian TARGET, OBJECTS, dan MOCS.

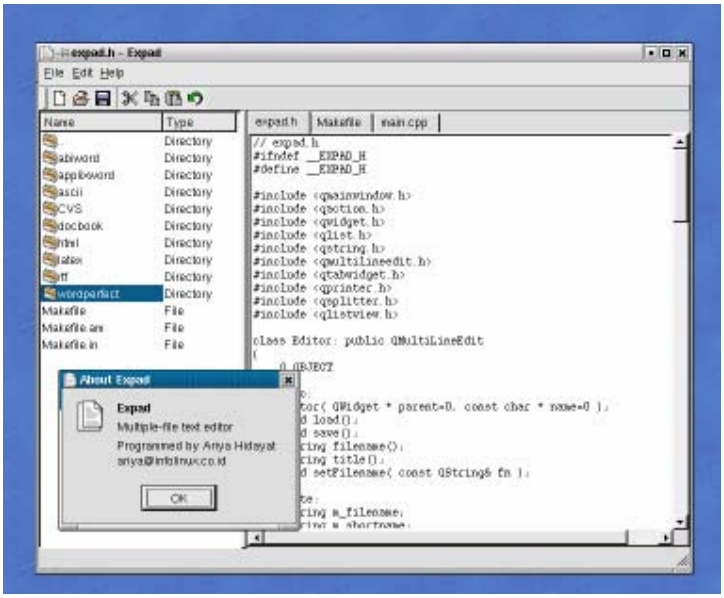
```
 1: QTDIR     = /usr/lib/qt2
 2: QTLIB     = $(QTDIR)/lib
 3: QTINCLUDE = $(QTDIR)/include
 4: QTMOC     = $(QTDIR)/bin/moc
 5:
 6: TARGET    = expad
 7: OBJECTS   = expad.o main.o
 8: MOCS      = expad.moc
 9:
10: CC        = g++
11: CFLAGS    = -Wall -O2
12: LIBS      = -lqt
13:
14: $(TARGET): $(MOCS) $(OBJECTS)
15:    $(CC) $(LFLAGS) -o $(TARGET) $(OBJECTS) $(LIBS) -L$(QTLIB)
-I$(QTINCLUDE)
16:
17: %.o: %.cpp
18:    $(CC) $(CFLAGS) -I$(QTINCLUDE) -c $< -o $@
19:
20: %.moc: %.h
21:    $(QTMOC) $< -o $@
22:
23: clean:
24:    rm $(OBJECTS) $(TARGET) $(MOCS)
```

Listing 4. Makefile

Begitu Anda sudah mendapatkan semua file yang dibutuhkan, yaitu `expad.h`, `expad.cpp`, `icons.h`, `main.cpp`, dan `Makefile`, maka kompilasi bisa dilakukan dengan perintah:

```
make
```

Bila semuanya lancar dan tidak ada sesuatu yang salah, Anda akan mendapatkan file executable `expad` yang siap digunakan. Selamat karena baru saja Anda menghasilkan editor yang sudah lebih baik dari Notepadnya Windows !



Gambar 1. Expad sedang beraksi

Sejenak menggunakan dan mengamati kerja Expad ini (Gambar 1), bisa disimpulkan bahwa spesifikasi seperti yang dibahas di awal tulisan ini sudah tercapai. Namun demikian, tentunya ada beberapa penyempurnaan yang masih bisa dilakukan. Anda dapat menganggapnya sebagai latihan kecil guna lebih mengenal seluk-beluk Qt.

Expad punya fasilitas *Close All*. Dengan teknik implementasi yang sama, mestinya juga tidak susah untuk mewujudkan fasilitas *Save All*. Sementara itu, bisa Anda saksikan bahwa font yang digunakan di widget Editor selalu sama, yakni adobe-courier. Pilihan yang lebih baik adalah menyiapkan kotak dialog konfigurasi yang salah satunya memungkinkan user mengganti font yang digunakan, bahkan juga tata warnanya.

Di sisi lain, DirList bisa mendapatkan sedikit perbaikan. Kalau untuk direktori ada icon bergambar folder, mengapa tidak ada icon serupa untuk item yang berupa file ? Anda bahkan bisa membedakan icon yang digunakan untuk file teks (berekstensi txt), file README, file gambar, dan jenis-jenis lainnya. Sebuah direktori yang tidak bisa dibaca

(alias tidak *readable*) seharusnya juga mendapatkan icon yang berbeda, misalnya gambar folder yang terkunci.

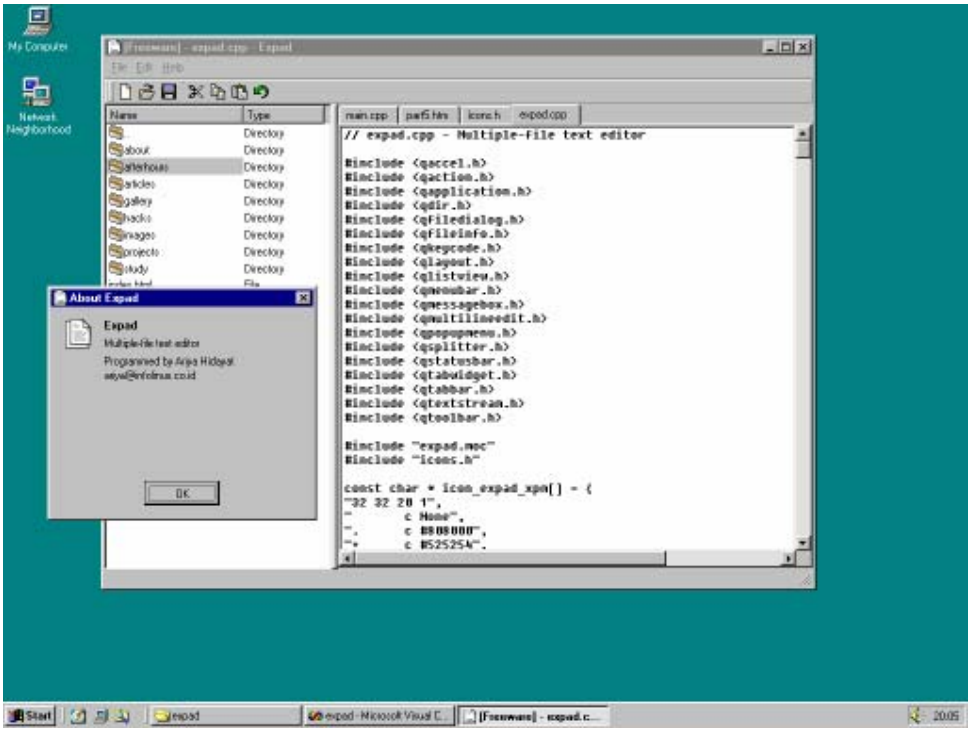
Modifikasi-modifikasi kecil di atas bisa dilakukan dengan hanya menambah beberapa baris program di sana-sini tanpa harus merombak strukturnya secara keseluruhan.

Expad for Windows ?

Ketika mengawali seri pemrograman ini disebutkan bahwa Qt adalah toolkit yang bersifat portabel. Maknanya adalah bahwa aplikasi-aplikasi yang dibuat sepenuhnya dengan kelas-kelas yang disediakan oleh Qt benar-benar bersifat *cross-platform*, bisa dikompilasi di sistem operasi yang didukung Qt, yaitu aneka varian Unix (termasuk Linux) dan Windows. Sifat *cross-platform* ini menyebabkan source-code program sama sekali tidak perlu diubah manakala dilakukan *porting* ke sistem operasi lain. Dengan demikian Anda bisa mendesain program tertentu di Linux, membawa source-codenya ke Windows, kemudian melakukan kompilasi ulang dan menghasilkan program baru yang sudah *for Windows*.

Sayangnya, Qt versi Windows tidaklah diberikan secara cuma-cuma dan juga tidak open-source. Anda harus membayar untuk mendapatkan lisensi pengembangan aplikasi Windows dengan Qt. Berita baiknya adalah bahwa Trolltech - si pembuat Qt - membuat perkecualian bahwa Qt/Windows ini tersedia juga dalam edisi non-komersilnya. Dengan menggunakan library Qt yang disediakan dalam edisi ini, Anda bisa menghasilkan aplikasi Windows sepanjang aplikasi tersebut juga sifatnya *free-software*.

Memanfaatkan edisi non-komersil dari Qt/Windows yang bisa didownload di www.trolltech.com/developer/download/qt-win-noncomm.html, Expad juga dapat dibuat untuk Windows. Karena library yang disediakan hanya cocok untuk Microsoft Visual C++, mau tidak mau Anda harus menggunakannya jika ingin mengkompilasi Expad di Windows. Dengan portabilitas Qt, sama sekali tidak ada file source-code yang perlu diubah, cukup sesuaikan proses kompilasinya (menggunakan Project di Visual Studio). Hasilnya adalah *Expad for Windows*, berpenampilan dan berperilaku benar-benar serupa dengan versi Linuxnya, seperti bisa disaksikan di Gambar 2 di bawah ini. Menarik, bukan ?



Gambar 2. Expad di Microsoft Windows

Dari studi kasus pengembangan editor multifile ini bisa Anda saksikan sendiri bagaimana kekayaan API dari Qt menjadi sangat menolong. Berbekal desain dan rancang bangun untuk aplikasi yang akan dikembangkan, pekerjaan selanjutnya nyaris seperti merangkaikan blok-blok siap pakai yang sudah disediakan Qt. Cepat dan nyaman. Kalau Anda sempat mengenal toolkit yang lain, misalnya GTK atau wxWindows, mengkonstruksi editor seperti Expad (dengan fitur-fitur yang setara) akan menjadi berbeda dan susah untuk tidak melampaui batas 1000 baris program. Begitu program selesai, dengan sifatnya yang portabel, Anda bisa mengkompilasinya di Windows untuk mendapatkan versi Windows program tersebut dengan mudah dan cepat.

Tanya Jawab

Tanya: Pada saat menjalankan `make`, muncul pesan kesalahan *icons.h: No such file or directory*. Bagaimana ?
Jawab: Ini terjadi karena file `expad.cpp` membutuhkan file header `icons.h` (untuk pixmap di toolbar), tetapi file header ini tidak ditemukan. Sebagaimana sudah disebutkan, Anda bisa mengambil file `icons.h` dari bagian kedua seri tulisan ini.

Tanya: Proses compile berjalan lancar. Namun pada saat Expad dijalankan muncul pesan kesalahan "Segmentation Fault". Mengapa ?
Jawab: Penyebab paling utama biasanya adalah ketidaksinkronan antara file hasil MOC dengan file header. Solusi: jalankan `make clean` dahulu sebelum `make`.

Tanya: Saat Expad dijalankan, saya ingin struktur direktori yang disajikan dimulai dari *home directory*, bukan dari *root directory*. Bagaimana mengubahnya ?
Jawab: Lihatlah konstruktor untuk kelas DirList, tepatnya di baris 213. Perintah `setDir( "/" )` harus diubah, tidak lagi menunjuk ke / (*root directory*), tetapi ke *home directory*. Bagaimana mengambil *home directory* ? Anda bisa menggunakan fungsi statik `QDir::homeDirPath()`. Lihat juga referensi Qt tentang kelas QDir untuk lebih jelasnya.

Tanya: Adakah cara untuk mengganti icon program ?
Jawab: Tentu. Icon program didefinisikan oleh array `icon_expad_xpm`, lihat file `expad.cpp` dari baris 26. Untuk menggantinya, lakukan langkah-langkah berikut. Cari dahulu icon pengganti, sebaiknya berukuran 32x32 piksel saja. Ubah menjadi format XPM (X Pixmap), misalnya dengan program pengolah citra seperti Gimp. Selanjutnya tinggal *copy-and-paste* isi file XPM tersebut menggantikan `icon_expad_xpm`. Bisa gunakan editor bisa (bahkan Expad sekalipun!) karena format XPM sebenarnya adalah file teks biasa.

Tanya: Tampilan struktur direktori menggunakan icon folder yang tidak begitu saya sukai. Bagaimana menggantinya ?
Jawab: Mirip dengan yang jawaban dari pertanyaan sebelumnya. Di sini yang musti Anda modifikasi adalah `icon_folder_xpm`, masih di file `expad.cpp` mulai baris 81.

Tanya: Bisakah saya mengedit file teks yang berukuran besar, misalnya beberapa MB, dengan Expad ?
Jawab: Berbeda dengan Notepadnya Windows, widget QMultiLineEdit bisa menangani teks sebanyak-banyaknya, hanya dibatasi oleh kapasitas memori yang ada. Karena Expad menggunakan widget ini maka dengan sendirinya Expad akan sanggup mengurusi file yang ukurannya besar. Hanya saja kemungkinan Anda akan mengalami gangguan performansi seperti waktu loading yang lambat, pergerakan kursor yang patah-patah, dan lain-lain.

Tanya: Jika saya mengutak-atik hingga tidak ada file yang dibuka oleh Expad, maka tiba-tiba setelahnya Expad berperilaku agak aneh. Kenapa bisa demikian ?
Jawab: Ini adalah masalah Qt karena kelas QTabWidget tidak dirancang untuk *tidak* mempunyai tab sama sekali. Untuk mengatasinya, Anda bisa memodifikasi program sehingga jika hanya tinggal satu file yang aktif (berarti hanya satu tab di QTabWidget), maka perintah Close bukan berarti menutup file tersebut, tetapi hanya menggantinya dengan editor baru saja.

Tanya: Proses compile berjalan lancar. Namun pada saat Expad dijalankan muncul pesan kesalahan "Segmentation Fault". Mengapa ?
Jawab: Penyebab paling utama biasanya adalah ketidaksinkronan antara file hasil MOC dengan file header. Solusinya gampang: jalankan `make clean` terlebih dahulu sebelum `make`.

Tanya: Sering muncul warning seperti ini: *const char * configure_xpm[27] defined but not used* saat mengkompilasi Expad. Apakah ini aman?
Jawab: Ini terjadi karena beberapa definisi pixmap di file header `icons.h` tidak digunakan semua. Bisa Anda lihat, toolbar Expad ini lebih sederhana dibandingkan

program contoh yang pernah dimuat di edisi kedua, bukan ? Lepas dari masalah itu, warning tersebut bisa diabaikan saja.

Tanya: Apakah source code untuk Expad juga diedit dengan Expad ?
Jawab: Tentu saja tidak. Hal ini serupa dengan lingkaran ayam dan telur. Bagaimana Expad bisa digunakan kalau source codenya sedang dibuat ?

Tanya: Bagaimana membubuhkan kemampuan *color syntax highlight* seperti yang lazim dijumpai di editor yang canggih ?
Jawab: Untuk ini Anda harus merancang kelas Editor yang digunakan, tidak bisa langsung diturunkan dari QMultiLineEdit begitu saja. Walaupun ini pekerjaan berat namun bukan berarti mustahil untuk dilakukan. Silakan mengambil contoh dari program lain yang punya fasilitas serupa seperti Freddy (freddy.sourceforge.net) atau Kate (kate.sourceforge.net).

Tanya: Saya sudah berhasil pula melakukan kompilasi Expad di Windows akan tetapi mengapa di titlebar selalu ada imbuhan *[Freeware]* ?
Jawab: Hal ini sebuah keterbatasan karena digunakan Qt Windows edisi non-komersial (QT-NC). Jika Anda ingin menghilangkannya silakan gunakan Qt Windows yang lisensi penuh (dan membayar lebih dari USD 1500 untuk itu). Untuk lebih rincinya, lihatlah FAQ mengenai ikhwal ini di www.trolltech.com/developer/faq/noncomm.html.

Sejarah KDE

Ariya Hidayat (ariya@kde.org, http://ariya.pandu.org).

KDE (K Desktop Environment), sebuah lingkungan desktop yang terkenal stabil, mudah digunakan, dan cocok untuk pemula adalah proyek open-sourcebesar dengan 800 lebih kontributor dan 2,6 juta baris program. KDEtidaklah serta merta muncul begitu saja seperti mainan seorang pesulap.Sejak kelahirannya Oktober 1996, proyek KDE sudah melewati masa balitanyadengan peluh dan segala kericuhan. Tulisan ini mencobamengangkat beberapa sisi lain dari hikayat perjalanan KDE, mulai darikelahiran hinggaperkembangannya.

Dari Tubingen ke Penjuru Dunia

Matthias Ettrich adalah seorang mahasiswa Universitas Tubingen, Jerman yang di pertengahan tahun90-an dikenal sebagai programer LyX. Ketika masa-masa penggunaan TeX sebagai *typesetter* yang profesional merambah ke mahasiswa, Matthias melihat bahwa ada hal yang masih lowong dengan LaTeX: sebuah antarmuka grafis. Tidak ada yang menyangkal bahwa cetakan hasil typesetting LaTeX sungguh berkualitas, akan tetapi banyakpula yang mengeluhkan susahnya menggunakan LaTeX sehari-hari. Memang, bagi kebanyakan orang menjalankan perintah baris yang penuh dengan trik sana-sini lebih mirip dengan pekerjaan seorang programer tatkala melakukan compileprogramnya.

LyX adalah aplikasi kecil yang akhirnya menjadi 'bayi' kesayangan Matthias. Apayang ditawarkan LyX sesungguhnya relatif sederhana: kemudahan menyusun dokumen dengan tampilan grafis yang menggunakan menu dan fitur ala WYSIWYG (*What YouSee Is What You Get*). Namun, sebenarnya sendiri LyX adalah front-end untuk LaTeX. Dengan kata lain, LyXtetap memanfaatkan LaTeX manakala akan mencetak dokumen. Hasil akhirnya,seorang user bisa menikmati kecanggihan LaTeX namundengan kemudahanantarmuka grafis.

Bersyukurlah bahwa Matthias Ettrich tidak berhenti hanya dengan LyX. Di satu masasekitar musim gugurtahun 1996, ia mengirimkan e-mail ke mailing-list LyX:

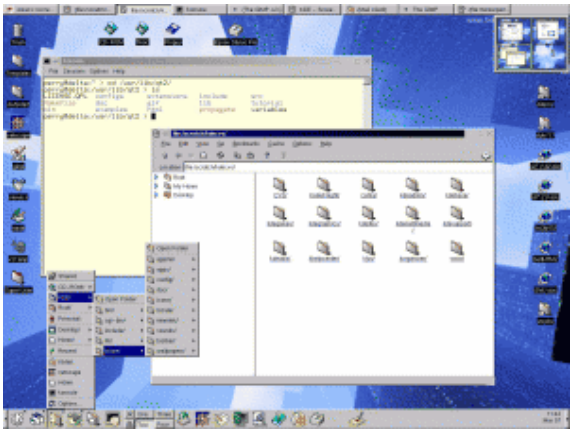
```
To: lyx@via.ecp.fr
Subject: Kool Desktop Environment
From: ettrich@peanuts.informatik.uni-tuebingen.de
Date: Mon, 14 Oct 1996 15:19:00 +0100 (MET)
Reply-To: lyx@zen.via.ecp.fr
Sender: owner-lyx@zen.via.ecp.fr
Hello,

I try to start another project with the goal to make Unix/X11 more
userfriendly. I just want to start this and do little coding, most
of my programming time will still go into the LyX development.
```

yang diikuti dengan penjelasan panjang lebar mengenai visinya sendiri tentang *KoolDesktop Environment*, proyek yang sasarannya adalah membuat Unix menjadi ramah dan mudah digunakan (*user-friendly*). Belakangan nama Kool

DesktopEnvironment ini diringkaskan menjadi KDE saja, dan malahan akhirnya K pada KDE tidak lagi berasosiasi dengan *Kool.*

Karena tujuannya yang relatif ambisius, tentu saja makan waktu cukup lama bagi tim KDE (yang dikepalai oleh Matthias) untuk menelurkan versi pertama dari KDE.Apalagi, karena betul-betul dibangun dari awal, maka terdapatbanyak sekali pekerjaan-pekerjaan berat (yang sifatnya membangun fondasi dasar) yang harusdikerjakan dengan sungguh-sungguh. Tim ini juga memutuskan untuk menggunakanberbagai lisensi perangkat lunak *open-source* untuk KDE dan infrastrukturnya.

Setelah penuh perjuangan mendesain, memprogram, dan menyusun berbagai komponen yang diperlukan bagi terwujudnya sebuah lingkungan desktop, makamusim panas tahun1998 menandai dilepasnya KDE 1.0, versi stabil pertamayang dapat diinstalasi dan dinikmati oleh pengguna Linux. Tidak hanya itu,KDEjuga tersedia untuk varian Unix yang lain, antara lain FreeBSD,HPUX, dan Sun Solaris.



Tampilan KDE 1.0

KDE versi pertama ini terdiri atas beberapa perangkat dasar. Sesuai namanya, yang utama ditawarkan KDE adalah lingkungan desktop yang modern,mencakup sistem menu (*KStart*), panel seperti taskbar-nya Windows(*KPanel*), serta window manager (*KWM*). Sebagai catatan,sungguhpun KDE mempunyai window managernya sendiri, yakni KWM, tidak adamasalah jika KDE digunakan bersama window manager lain seperti *WindowMaker* atau *Enlightment*.

Di samping lingkungan desktop, KDE juga menawarkan sejumlah aplikasi yang sangat bermanfaat. Ada *file manager* untuk mengurusi file dan harddisk, *ControlCenter* guna mengatur berbagai konfigurasi sistem (*KControl*), *imageviewer* untuk melongok isi file gambar , terminal emulator bernama Konsole, e-mail client (*KMail*), dialer (*kppp*), organizer (*KOrganizer*) dan puluhan aplikasi lainnya. Selain memiliki konsistensi tampilan (dalam soal menu, ikon, dantata warna), kesemua aplikasi ini juga saling terintegrasi dengan baik.Sebagai contoh, mengklik file PNG di *file manager* akan otomatik mengeksekusi *image viewer*sehingga isi file PNG tersebut bisa dilihat dengan mudah.

Rendah Hati, Bertangan Dingin

Banyak orang bilang bahwa programer, hacker, developer, maupun jagoan komputer biasanya congkak danbanyak omong. Nah, Matthias Ettrich adalah kebalikan dari semua itu. Mengenang perjumpaan penulis dengan Matthias sekitar dua tahun yang silamdi Karlsruhe, mudah untuk diamati bahwa memang kesan penampilan yangramah dan menarik langsung tersiratkan dalam dirinya. Matthias juga samasekalitidak sombong dan malahan cenderung pemalu sehingga bahkan banyakyang tidak tahu (dan tidak menyangka) bahwa LyX dan KDE lahir dari tangan dinginnya. Setelah menyelesaikan studi di UniversitasTubingen, kini Matthias bekerja di Trolltech untuk terus memoles Qt menjadi toolkit yangprofesional.

Keseragaman dan kerjasama yang baik antar aplikasi KDE dimungkinkan karena kesemuanya dibangun dengan menggunakan pustaka standar KDE, yang lazim disebut sebagai *kdelibs*.Pustaka ini mencakup kumpulan rutin yangdibutuhkan untuk pemrograman aplikasi grafis, fitur network, dan interaksi dengan pengguna. Kehadiran pustaka seperti ini telah dinanti-nanti oleh banyak orang, terutama programer yang berkehendak membangun aplikasi berbasis grafis dengan mudah dan nyaman.

Karena menawarkan sesuatu yang baru dan memikat, KDE 1 cepat sekali menarik perhatian orang. Tercatat SuSE, sebuah distro yang bermarkas di Jerman, menyertakan KDE 1 pada SuSE versi 6.0. Di sisi lain, meski Red Hatsaatitu belum melirik KDE, instalasi KDE 1.0 dalam bentuk paket RPM tidakresmi plus petunjuk instalasinya untuk Red Hat 5.2 telah tersedia danlangsung dicoba oleh banyak orang. Pun tersedia paket-paket serupa untukdistro lain.

Di sisi lain, kesuksesan KDE 1 tidak membuat tim programer KDE (yang akhirnya kian bertambah) lantas ongkang-ongkang kaki. Melalui serangkaiankerja keras yang penuh semangat, versi perombakan pertama yang dirilissebagai KDE 1.1 hadirbulan Februari 1999, yang disusul dengan perbaikankecil (KDE 1.1.1) bulan Mei dalam tahun yang sama. Tidak kurang, berbagaimedia pun mulai meliput kehadiran sang pendatang baru ini, mulai darimajalah LinuxJournal, eksebisi LinuxWordExpo, hingga arena CeBIT 1999 diHannover. Walhasil, beberapa penghargaan juga berhasil diraih,antara lain LinuxWorld Editor Choice Award 1999, "SoftwareInnovation of theYear" CeBIT 1999, dan Linux Journal 1999 Readers'Choice.

K = Kontroversi ?

Bahagiakah semuanya dengan kedatangan KDE ? Ternyata tidak. Beberapa orang, termasuk **RichardStallman**, yang dengan keras menyuarakan kampanye Free Software menolak kehadiran KDE. Hal ini bukan karena KDE itusendiri yang nyata-nyata merupakan perangkat lunak yang bebas dan *open-source*. Alasan utamanya adalah bahwa mereka tidak merasa nyaman karena KDE dibangun dengan *toolkit* bernama Qt, keluaran dari Trolltech AS, sebuah perusahaan software di Oslo, Norwegia. Hingga saat itu, Qt bukanlah merupakan pustaka yang bersifat *open-source*. Walhasil, meski KDE benar-benar tergolong *open-source*, kehadirannya sendiri ditentukan oleh keberadaan Qt yang komersil. Di sisi lain, ada pula orang-orangyang tidak peduli dengan urusan lisensi seperti misalnya **Linus Torvalds** yang memberikan komentar:

```
My opinion on licenses is that "he who writes the code gets to
chose his license, and nobody else gets to complain". Anybody
complaining about a copyright license is a whiner.
```

Persoalan Qt ini meluas sehingga akhirnya muncullah berbagai gagasan untuk mengatasi permasalahan ini. Yang paling sederhana tentu adalah dengan membuat Qt menjadisebuah produk open-source. Trolltech sendiri telah diminta oleh berbagai kalangan untuk melakukannya. Akan tetapi, karena model bisnis Trolltech tidakmengijinkan hal ini, maka keinginan tersebut tidak dapat diakomodasi.

Lahir pulalah sebuah proyek yang bernama *Harmony*. Tujuannya adalah menciptakan versi Qt yang open-source. Pengembangannya sendiri benar-benar di luar Trolltech dantidak bersangkut paut dengan Qt yang asli. Diharapkan, jika kelak proyek Harmony tuntas, maka semua perangkat lunak yang memanfaatkan Qt (jelas termasuk KDE) dapat dialihkan untuk menggunakan pustaka Harmony tanpa perlu mengubah programnya sedikit pun. Idenya kurang lebih serupa dengan Mesa3D yang merupakan implementasi open-source dari OpenGL. Dalam perkembangannya, proyek Harmony tidak menarik bagi banyak developer sehingga mati perlahan-lahan dan resmi ditutup pada Januari 1999.

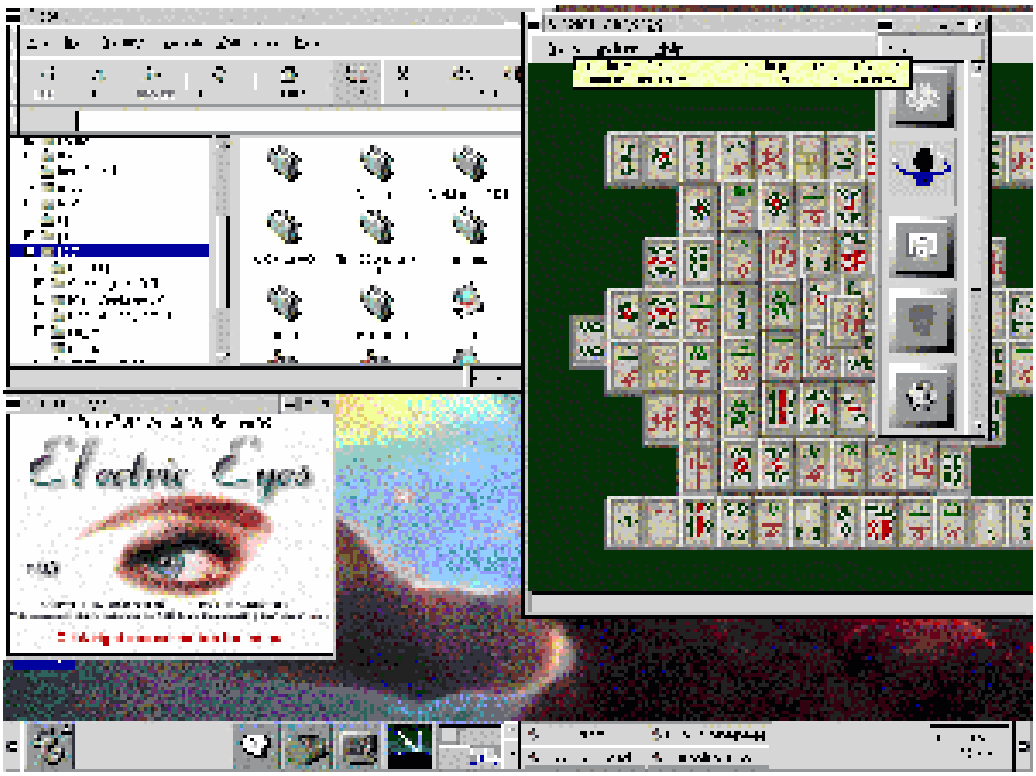
Titisan Viking

Norwegia, negara yang *mojok* di utara Eropa ini barangkali populer karena bangsa Vikingnya yang melegenda, sebut saja Eric Si Merah yang menguasai Greenland. Kisah kerakyatannya juga tidak kalah masyhur, *troll*alias monster kuno yang mengerikan (dan setidaknya dicukilkan di *Lord of TheRing* dan *Harry Potter*) disebut-sebut berasal dari daerah mereka. Di abad informasi seperti sekarang, Norwegia bagi kalangan pengamat software sering langsung terasosiasikan dengan *Trolltech*, sebuah perusahaan yang bermarkas di Oslo dan memproduksi tookit bernama Qt (perhatikan sama namaTrolltech yang masih mengandung *troll*). Tetapi tunggu dulu, dikota yang sama ada pula *Opera Software* yang menghasilkan Opera, browser yang mengklaim diri sebagai yang tercepat di muka bumi. Bukan kebetulan kalau versi Linux dari Opera dikembangkan menggunakan Qt. Selain berlandaskan justifikasi teknis, langkah demikian juga seakan ingin mempertontonkan kekompakan mereka, cucu-cucu bangsa Viking.

Pilihan lain yang akhirnya sukses adalah mengembangkan lingkungan desktopalternatif yang sejajar dan dapat menggantikan KDE. Inilah yang dirintisoleh **Miguel de Icaza** dengan proyek GNOME (*GNU Network ObjectModel Environment*). Diluncurkan Agustus1997, GNOME dirancang untuk menjadi produk yang benar-benar sesuai standar *Free Software Foundation*. Karenanya, toolkit grafis yang dijadikan pendukung GNOME adalah GTK+, yakni sebuah pustaka yang sebelumnya dikenal sebagai fondasibagi GIMP (*GNU Image Manipulation Program*), yang benar-benar*free software* (dengan lisensi LGPL).

Kehadiran proyek GNOME dan berbagai tekanan-tekanan dari para advokat perangkat lunak akhirnya mencairkan sikap Trolltech. Akhirtahun 1998, mereka akhirnya memutuskan untuk melepaskan versi Qt untuk Unix/X11 dengan QPL (*Qt PublicLicense*) yang menjadikanQt sebagai produk open-source. Sekitar dua tahun kemudian, Trolltech juga menambahkan pilihan kemungkinan lisensi dengan GNU GPL(*General Public License*)versi 2.

Tentu saja, Qt yang menjadi open-source tidak menyebabkan proyek GNOME berhenti.Karenasudah terlanjur jalan, maka melewati masa-masa awal pengembanganyangberliku-liku, tim GNOME dengan komandan Miguel de Icaza akhirnyamelepas GNOME1.0 pada bulan Maret 1999. Rilis yang dilakukan diLinuxWord Expo ini gaungnyamendunia dan memikat berbagai pihak yangmencari alternatif dari KDE.



GNOME yang menjadi pesaing KDE

Persaingan antara KDE dan GNOME semakin diperpanas ketika Red Hat mengemas GNOME sebagaipilihan default. Linux Mandrake, sebuah distro yangsekarangdikenal *user-friendly*, mulanya sendiri lahir sebagai varian Red Hat tetapi menggunakan KDE sebagai default, versi paling awalnya keluar di bulan yang sama dengan dirilisnya KDE1. Sementara itu, SuSE tetap setia dengan KDE bahkan hingga saat sekarang.

Banyak pihak yang melihat bahwa perseteruan KDE vs GNOME sebagai hal yangburuk. Jikalah kedua tim bergabung, maka tentu hasilnya akan lebih dahsyat dan tidak akan energi yang terbuang percuma. Di sisi lain, justruadanya persaingan seperti ini akan memancing kemunculan inovasi yangmembuat masing-masing kubu menjadi semakin kompetitif. Sementara itu,secara teknis sendiri tim developer KDE dan GNOME menggunakan pendekatandan teknologi yang berbeda, dan sudah barang tentu akan memakan waktu dantenaga untuk saling diselaraskan.

Bagi kebanyakan pemakai komputer, baik KDE ataupun GNOME bisa saja dianggapsebagai berkah. Toh, kelebihan penggunaan *free software* adalahadanya

kebebasan penuh. Yang tidak suka KDE tinggal beralih ke GNOME,sementara itu yang merasa seleranya tidak pas dengan GNOME bisa menggunakan KDE. Benar-benar pilihan yang bebas !

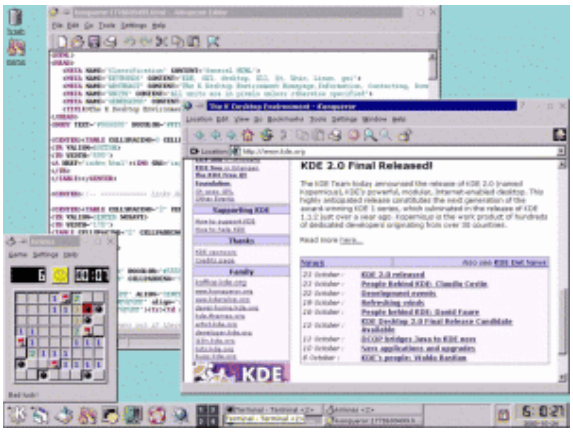
Setelah edisi pertama lepas, para developer KDE yang jumlahnya sedikit demisedikit bertambah memfokuskan diri untuk KDE 2. Berbeda dengan KDE 1,maka KDE 2 akandibangun berlandaskan Qt 2. Dibandingkan pendahulunya,versi kedua dari Qt inimempunyai banyak fitur baru seperti misalnyadukungan Unicode, theme, XML, danbanyak hal lagi.

KDE 2: Serasa Terlahir Kembali

Mengingat beberapa bagian utama dari Qt 2 sama sekali berbeda dengan Qt 1, mautidak mau KDE juga mengalami perombakan total. Tidak salah jika dikatakanbahwa KDE 2 adalah versi yang benar-benar baru, berbeda total dengan KDE 1.Tentu saja, pengembangannya sendiri tetap menyandang tujuan yang tetapsama: menghadirkan antarmuka grafisyang nyaman dan mudah digunakan.

Beberapa bagian internal KDE juga harus ditulis ulang, baik karena desain awalyang salah maupun demi efisiensi yang lebih baik. Sebagai contoh, panel di KDE 1 telah menjelma menjadi Kicker di KDE 2 yang lebih tangkas danmenarik. Teknologi pendukung seperti KIO pun dibangun untuk memudahkan akses data dengan mudah keberbagai device, seperti disket, drivejaringan, share pada Windows SMB,d an bahkan akses data via protokolseperti HTTP dan FTP.

Sadar bahwa Internet adalah tren masa depan, tim KDE juga merancang lingkungan desktop yang bersifat *Internet-ready*. Walhasil, dimulailah polesan yanglebih baik terhadap KMail dan KNode yang menjadi programfavorit untuk akses e-mail dan newsgroup. Penggunaan KIO dengan baik juga menghasilkan *network-transparency*yang sangat mendukung penggunaan sebuah desktop oleh pemakainya. Sebagai ilustrasi, saat mengedit sebuahfile, akan sama saja prosedurnya apakah file tersebut bercoko ldiharddisk lokal maupun tersimpan server FTP yang remote. Tidak perlu repot-repot upload secara manual.



Generasi kedua KDE

Beberapa programer juga menyadari pentingnya keberadaan sebuah browser web yang bisa diandalkan. Microsoft mengemas Internet Explorer sebagai bagian dari Windows, jadi mengapa hal yang sama tidak bisa dilakukan untuk KDE ? Mengingat ketika sekitar tahun 1999 Mozilla baru dilepas dari Netscape dan terbukti sama sekali tidak stabil, tim KDE memutuskan untuk mengembangkan sendiri *engine* untuk membangun sebuah browser web yang mendukung penuh HTML 4 dan CSS (yang akhirnya dibaptis sebagai KHTML). Aplikasi baru yang bernama Konqueror muncul ke dunia, sebuah file manager yang juga browser web (yang pastilah memanfaatkan KHTML).

Seiring dengan semakin stabilnya fondasi lingkungan desktop yang dibangun oleh berbagai*library* KDE, maka sekelompok programer KDE juga mulai memikirkan untuk mengembangkan tambahan aplikasi KDE yang lebih serius. Karena dunia Linux miskin dengan IDE(*Integrated Development Environmen*t) yang notebene sangat berfaedah untuk para developer, maka hadirlah KDevelop, sebuah IDE modern yang berbasis KDE. Dikemas bersama KDE 2, debut pertama KDevelop (juga dilengkapi *debugger*yang terintegrasi) langsung membuat gebrakan yang manis dan lekas menjadi pilihan banyak programer tatkala ingin mengembangkan aplikasi grafis untuk Linux.

Sebagai sang pendahulu, KDE 1 sering dicemooh terlalu mencoba tampil mirip dengan Microsoft Windows. Melalu dukungan *widget style* yang ada ditawarkan Qt2, maka KDE 2 tidak lagi dapat disebut sebagai tiruannya Microsoft Windows.Dengan berbagai pilihan style yang ada, sebuah desktop yang menggunakan KDE 2 bisa saja dibuat serupa dengan Mac, Windows, BeOS, Motif, atau berbagai ragam pola tampilan yang lain. Kalau sebelumnya KDE 1 masih mentoleransi user dengan keterbatasan warna pada monitornya, maka seiring dengan meluasnya kartu grafisyang menghadirkan kekayaan *True Color*, maka KDE 2 juga menghadirkan ikon dan tata warna yang lebih kaya akan nuansa.

Kira-kira setahun menjelang rilis, diselenggarakan pula KDE Two, sebuah hackfest di Erlangen, Jerman yang tidak hanya untuk mengikuti tradisi KDE 1.x, tetapi juga menjadi ajang tukar pikiran sejumlah developer inti KDE yang memang kebanyakan berdomisili di Eropa.

Belakangan terbukti bahwa menulis ulang KDE sehingga dapat memanfaatkan sepenuhnya kekuatan Qt 2 memakan waktu dan energi yang besar. Berbeda dengan KDE1.x yang relatif sederhana, KDE 2 adalah proyek yang lebih serius. Developernya juga semakin tidak main-main, sebelum akhirnya dirilis Oktober 2000, KDE 2 telah melewati 5 versi beta dan 1 RC (Release Candidate). Ketika itu adalah **Waldo Bastian** (yang bekerjadi SuSE) bertindak selaku sang *release coordinator*.

Karena meningkatnya dukungan Unicode, maka KDE mulai dapat digunakan dinegara-negara yang menggunakan aksara non Latin, seperti misalnya Arab, Jepang, Korea, Rusia,Cina, dan banyak lagi. Memang, standar Unicode mengijinkan karakter yang lebih universal (yang tidak akan tertampung oleh himpunan karakter ASCII semata). Hal ini juga berarti secara langsung basis pengguna KDE meningkat drastis, tidak melulu mereka yang memanfaatkan huruf-huruf Latin saja.

Sekali lagi, kesuksesan rilis KDE 2 juga menyebabkan beberapa perhargaan berhasil disabet, antara lain Show Favorite dan Linux Community Award diajang LinuxWorld Expo 2000, Frankfurt, baik Editor's Choice 2000 maupun Reader's Choice 2000 dari LinuxJournal, Best Open Source Project di LinuxWord Expo 2001 San Francisco, Reader's Choice Award dari LinuxJournal 2001, dan masih banyak lagi.

Beberapa versi perbaikan yaitu KDE 2.1 dan KDE 2.2 juga menyusul rilis KDE 2.0. Selain perbaikan bug, terdapat juga penambahan fitur dan optimasi di versi-versi tersebut, antara lain dengan hadirnya KPersonalizer untukmelakukan kustomisasi desktop dengan cepat, restrukturisasi Control Center untuk kemudahan, serta peningkatan kualitas Konqueror.

Tonggak Sejarah KDE
Oktober 1996: KDE lahir dari inisiatif Matthias Ettrich
Agustus 1997: KDE-One, hackfest pertama di Arnsberg (Jerman)
Juli 1998: Versi pertama, KDE 1.0 dirilis setelah 4 versi Betanya diujikan dalam 7 bulan terakhir
Februari 1999: KDE 1.1, versi perbaikan dari KDE 1.0
Oktober 1999: KDE-Two, sebuah acara hack-fest diselenggarakan di Erlangen (Jerman)
September 2000: Trolltech mengumumkan bahwa Qt resmi dilisensi menggunakan GNU GPL (General Public License)
Oktober 2000: KDE 2.0 dilrilis (setelah melalui 5 versi Beta dan satu Release Candidate)
Februari 2001: KDE 2.1
Agustus 2002: KDE 2.2
April 2002: Generasi ketiga, KDE 3.0 dirilis
Maret 2003: KDE 3.1: stabil, aman, indah

Teknologi KParts untuk Konqueror dan KOffice

Yang namanya teknologi komponen ternyata mulai menjamur. Berbagai pihak yang berkepentingan dengan pengembangan software hampir tidak melewatkan kesempatanuntuk menanamkan investasi dalam penelitian di bidang ini. Bisa dimengerti, dengan membuat sebuah aplikasi terdiri atas jalinan berbagai komponen yang saling berinteraksi, hal tersebut akan meningkatkan modularitasnya. Belum lagi, kemudahan mengembangkan satu komponen tanpa terlalu bergantung kepada komponen yang lain.

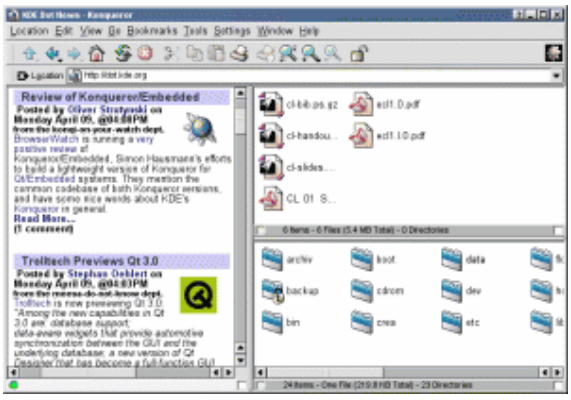
Tercatat Microsoft-lah yang gigih di garda terdepannya dengan teknologi COM (ComponentObject Model) yang belakangan diimplementasikan dalam OLE (Object Linking andEmbedding) dan ActiveX. Sederhananya, tatkala Anda menyisipkan sebuah laporan keuangan Microsoft Excel dalam dokumen Microsoft Word, maka secara tanpa sadar Anda sudah menjadi "korban" dari teknologi komponen ala Microsoft.

KDE pun tidak mau ketinggalan. Dipelopori oleh **Torben Weis**, dicanangkanlah proyek kecil untuk menghasilkan teknologi komponen di KDE yang dinamakansebagai OpenPart. Mula-mula OpenPart mengandalkan enkapsulasi interface dengan CORBA (Common Object Request Broker Architecture) yang relatif populer di dunia middleware. CORBA memang terbukti andal dan *network-ready*. Sayang, untuk penggunaan di dunia desktop, OpenPart yang berbasis CORBA terlalu lamban, bahkan dengan hingga batas

yang susah ditolerir. Belum lagi kompleksitasnya yang cukup tinggi, membuat hanya sedikit developer yang punya pengetahuan cukup untuk mengembangkan komponen dengan OpenParts.

Walhasil, terjadilah perombakan besar-besaran. OpenPart ditulis ulang dan direinkarnasi sebagai KParts. Terbukti, desain yang digagaskan dalam KParts terbilang sederhana dan mudah dipelajari namun menyimpan potensi yang lengkap untukmembangun komponen yang modern. Tidak hanya developer inti KDE seperti **David Faure** atau **Simon Hausmann** (yang memang terlibat langsung dipengembangan KParts), banyak programer KDE pun dapat mengambil manfaat dari teknologi KParts tanpa bersusah payah menyelami intisari CORBA terlebih dahulu.

Bila Anda sesehari menggunakan Konqueror, maka sesungguhnya Anda telah mendayagunakan kemampuan KParts. Sebagaimana telah disebutkan, KHTML adalah *engine*yang digunakan Konqueror dan diimplementasikan sebagai sebuah komponen KParts. Masih banyak lagi komponen lain, seperti misalnya untuk menampilkan citra, mengedit file teks, GhostScript dan sebagainya. Tidak heran jika Konqueror yangmendukung penuh KParts dapat bertindak sebagai *universal viewer* bagi file-file yang dikenali KDE (dan tersedia komponennya). Klik saja sebuah file PDF dalam Konqueror dan Anda akan dengan mudah melongok isinya. Hal yang sama untuk file teks, citra JPEG, PNG, PostScript, dan sebagainya.



Konqueror: browser, file manager, sekaligus universal viewer

KOffice, sebuah proyek ambisius yang bertujuan menghasilkan paket office terintegrasi untuk KDE, ternyata juga aplikasi yang dibangun berlandaskan KParts. Pada rilis awal KOffce 1.1, dengan **David Faure** yang saat itumasih bekerja untuk Mandrake berperan selaku release coordinator, terdapatbeberapa aplikasi seperti KWord (pengolah kata), KSpread (spreadsheet), KPresenter (presentasi), Kivio (diagram dan flowchart), serta KIllustrator (grafik vektor). Walaupun masih terbilang belia, KOffice sudah cukup fungsional dan memberikan harapan yang cerah di masa depan. Dengan berbasis KParts, akan sangat mudah bagi aplikasi-aplikasi KOffice untuk saling bertukar data, seperti misalnya menyisipkan lembar kerja KSpread di dalam dokumen teks di KWord.

Semakin Mantap dengan KDE 3

Kalau migrasi dari KDE 1 ke KDE 2 dirasakan sebagai perubahan yang drastis,makaberalih dari KDE 2 ke KDE 3 tidaklah demikian. Hal ini juga dipengaruhi kenyataan bahwa Qt 3 (yang menjadi fondasi KDE 3), secara hati-hati sudah dirancang agar tidak membawa perbedaan yang berarti dengan Qt 2. Karenanya,proses rilis KDE 3 sendiri lebih cepat dan tidak lagi berlarut-larut. Hanya sekitar setengah tahun lepas dari beredarnya KDE 2.2, KDE 3.0 sudah muncul kedunia, dengan **Dirk Müller** berperan sebagai *releasecoordinator*.

Mengingat waktu pengembangannya sendiri relatif singkat, tersisa banyak fitur yang diwariskan ke KDE 3. Walaupun demikian, perbaikan di sana sini (termasuk berbagai *bugfix*) tetap dilakukan seperti peningkatan kecepatan JavaScript dan Dynamic HTML untuk Konqueror, dukungan IMAP di KMail, style dan theme yang kian lengkap, dan banyak lagi. Yang menarik adalah bahwa KDE 3 lebih cepat dan lebih irit memori dibandingkan dengan versi sebelumnya, sesuatu hal yang jarang terjadi di dunia perangkat lunak komersil.

Sebuah paket aplikasi yang bernama KDE Edutainment juga memulai kiprah pertamanya berbarengan dengan KDE 3. Terdiri atas beberapa program "belajar sambil bermain", KDE Edutainment sangat tepat untuk anak-anak dan kaum remaja. Lihat saja koleksi aplikasinya, KTouch untuk latihan mengetik, KStar yang mensimulasikan peta langit, KVocTrain guna melatih kosakata, dan banyak lagi.

KDE 3 pun membawa tingkat konfigurasi ke arah yang heboh. Sebuah situs web www.kde-look.org muncul di tengah komunitas pengguna KDE yang menawarkankoleksi berbagai macam *goodies*, mulai dari style baru yang bisa menyulap KDE seindah MacOS, ikon-ikon bergaya Windows XP, tata warna alternatif, dan banyak lagi.

Dan, setelah nyaris setahun, barulah keluar KDE 3.1, sebuah perbaikan atasrilis KDE 3.0. Yang menyolok dari KDE 3.1 adalah karena waktu pengerjaannya cukup lama,bahkan hingga melewati tiga versi beta dan tujuh RC (Release Candidate). Satu hal yang sempat menunda rilisnya adalah dilakukannya *security audit*. Sebagaimana layaknya sebuahperangkat lunak yang dilepas ke publik, keamanan adalah isu yang mengingat.Tidak ada pengguna yang menginginkan lingkungan desktopnya mendadak di-crack oleh pihak-pihak jahil, hanya gara-gara *remote exploit* yang belum dibenahi.

KDE 3.1, yang paling terbaru dari KDE.

Tentu, ada juga imbuhan segudang fitur baru di KDE 3.1 seperti misalnyatampilanmodern dengan Keramik dan ikon Crystal, peningkatan kecepatan Konqueror, kompatibilitas yang lebih baik di KHTML, dukungan KMail S/MIME untuk attachment, fasilitas *desktop sharing* yang *built-in*, tambahan berbagai game, adanya download manager, dan banyak hal baru lainnya.

Perjalanan KDE yang telah melewati masa balitanya tidak membuat sekitar 800-an kontributor KDE lantas menjadi berpuas diri dan berhenti berinovasi. Rencana dan jadwal kegiatan sudah dibangun bersama-sama untuk generasi KDE berikutnya, termasuk KDE 3.2 yang kemungkinan hadir sekitar musim semi tahun ini.

Beberapa tambahan aplikasi akan ikut menyemarakkan KDE 3.2, seperti misalnya *Kopete* untuk instant messaging yang mendukung AIM, ICQ, Yahoo, dan banyak lagi atau *Umbrello*untuk pemodelan dengan UML. Melongok kesuksesan Ximian dengan Evolution-nya, tim KDE juga mencanangkan Kontact, aplikasi khusus PIM (Personal Information Management) yang mirip-mirip dengan Outlook. Dengan menggabungkan aplikasi mail, kalendar, dan dibubuhi dengan fitur groupware tentu diharapkan Kontact dapat menjadi pertimbangan bagi mereka yang ingin bermigrasi dariOutlook.

Di tengah hiruk-pikut kampanye .NET oleh Microsoft, segelintir orang mulai mengeksplorasibahasa C# (baca: C Sharp) yang diciptakan Microsoft khusus untuk platform.NET-nya tersebut. Mono, sebuah program yang juga open-source, telah merintis jalan panjang yang memungkinkan aplikasi .NET dijalankan di Linux, jelas dengan menggunakan C#. Dalam kaitannya dengan KDE, binding dari C# ke Qt telah berhasil diwujudkan denganproyek Qt#. Bukan mustahil bahwa dalam waktu dekat KDE# juga akan lahir, yang berarti aplikasi KDE bisa dibangun dalam kerangka C#.

Generasi ketiga KDE dirumorkan akan berumur cukup panjang, bahkan mungkin hingga versi3.6. Setelahnya, ketika Windows XP dan MacOS X mulai terasa usang, barulah barangkali KDE 4 bakal lahir ke dunia. Kita nantikan bersama !

Membangun Aplikasi KDE dengan KDevelop

Linux adalah wahana pengembangan aplikasi yang memberikan banyak fasilitas. Selain tersedianya GCC sebagai salah satu *compiler* yang modern, lengkap, dan andal, Linux juga menawarkan ratusan pustaka (*library*) yang dapat dipergunakan untuk mempermudah sekaligus mempercepat pengembangan sebuah aplikasi. Karena itu, sudah selayaknya seorang programer dapat memanfaatkan pustaka-pustaka ini sembari memperluas cakrawala pengetahuannya.

KDE yang merupakan akronim dari *K Desktop Environment* lebih dikenal sebagai lingkungan kerja berbasis grafis. Selain menyediakan *window manager*, KDE juga beragam aplikasi dalam aneka bidang, mulai dari *file-manager*, browser, hingga program-program multimedia.

Yang tidak banyak disadari adalah bahwa KDE juga merupakan platform untuk pengembangan aplikasi. Mengapa demikian ? Tidak lain adalah karena bersama KDE terdapat juga pustaka-pustaka yang menjadi kerangka pembangun sebuah aplikasi (*application framework*). Setiap sistem Linux yang menginstalasi KDE dipastikan mempunyai pustaka-pustaka standar KDE, yang lumrahnya disebut sebagai *kdelibs*. Bilamana Anda ingin membuat aplikasi dengan cepat dan tidak mau dipusingkan dengan pengaturan antarmuka pemakai, akses berbagai fungsi sistem, pengolahan file, dan lain sebagainya, Anda bisa memanfaatkan pustaka standar KDE ini.

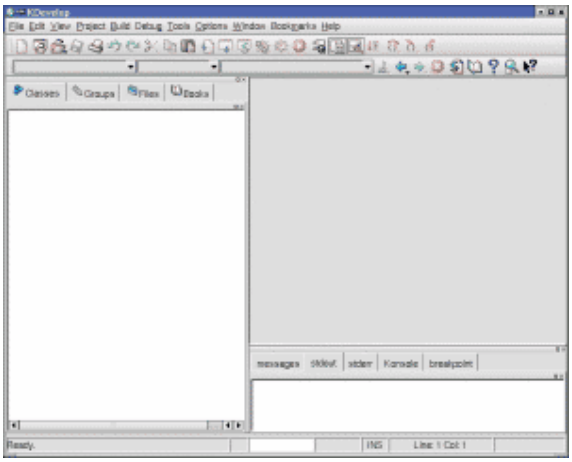
KDE sendiri sebetulnya mengandalkan diri pada pustaka Qt (dari Trolltech) sebagai fondasinya. Buat yang belum kenal, Qt dikenal sebagai toolkit untuk membangun aplikasi yang lintas platform di Unix/X11, Windows, Mac, dan sistem embedded. Akan tetapi, bagi sebagian besar developer KDE, bisa diklaim bahwa KDE bukanlah semata-mata pustaka toolkit sebagaimana Qt. Hal ini berarti bahwa KDE tidak melulu menyediakan setumpuk fungsi-fungsi yang bisa digunakan dalam beragam library-nya, tetapi juga menawarkan kerangka (*framework*) yang dapat dieksplorasi untuk menghasilkan aplikasi-aplikasi GUI yang modern. Lepas dari faktor kecakapan programmer yang menjadi salah satu penentu keberhasilan proses pengembangan sebuah aplikasi, kekayaan library yang dipunyai oleh KDE plus kelengkapan dokumentasi API-nya (yang bisa diakses dengan cuma-cuma) terbilang cukup menolong dan menjadikan penyusunan suatu aplikasi menjadi perangkaian balok-balok dasar yang memang sudah tersedia.

Tentu saja, kekayaan pustaka saja tidak akan cukup. Menyadari hal ini, tim KDE juga merilis sebuah Integrated Development Environment (IDE) bernama KDevelop yang bisa dipergunakan untuk membantu proses pengembangan aplikasi dengan lebih nyaman. Bagi yang telah familiar dengan Microsoft Visual Studio, KDevelop ini tidak akan terasa asing lagi. Fitur-fitur yang dimilikinya juga tidak kalah dengan Visual Studio, di antaranya adalah *project management*, editor canggih dengan *syntax highlight* dan *code completion*, debugger yang terintegrasi, *class parser*, akses ke dokumentasi Qt/KDE/C++ dengan cepat, penanganan configure/automake/autoconf, *wizard*, *class generator*, dan masih banyak lagi.

Jika sudah terinstal dengan baik, KDevelop bisa dijalankan dari KDE dengan memilih menu *K, Development, KDevelop*. Untuk pengguna Mandrake, menu ini bisa diakses dari *K, Applications, Development, Development environments, KDevelop*. Bila baru pertama kali ini Anda menjalankan KDevelop maka akan tampil aplikasi *KDevelop Setup* secara otomatis (Gambar 1). Program kecil ini sendiri mirip dengan sebuah wizard, jadi tinggal ikuti saja langkah demi langkahnya. Pada kebanyakan kasus, Anda hanya perlu mengklik tombol *Next* hingga langkah terakhir.

Gambar 1. KDevelop Setup

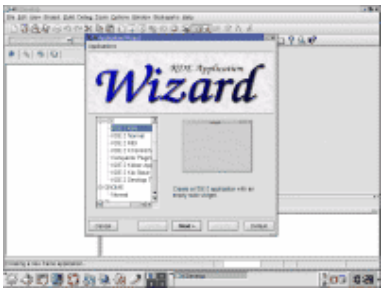
Selanjutnya Anda akan dihadapkan dengan KDevelop dalam keadaan awal (Gambar 2). Bisa Anda perhatikan bahwa penampilannya sangat serupa dengan Microsoft Visual Studio. Tepat di bawah *titlebar* adalah sederetan tombol-tombol *toolbar*. Sebagaimana umumnya toolbar, Anda cukup melewatkan pointer mouse di atas tombol-tombol ini untuk mengaktifkan *tooltip*. Bagian kiri terdiri atas sejumlah tab yang masing-masing ditandai dengan icon-icon tertentu, masing-masing untuk *Classes*, *Groups*, *Files*, dan *Books*.



Gambar 2. Tampilan awal KDevelop

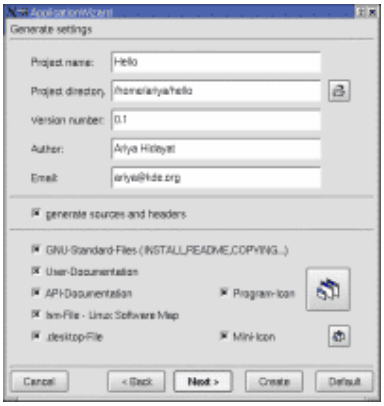
Supaya lebih jelas, Anda dapat mencoba membuat sebuah aplikasi KDE yang sangat sederhana. Untuk ini, pilihlah menu *Project, New* yang digunakan untuk mengaktifkan wizard pembuatan aplikasi baru (Gambar 3). Bisa dilihat bahwa wizard tersebut dapat dipergunakan untuk membuat berbagai macam aplikasi, dari aplikasi KDE, Qt, GNOME,

dan juga aplikasi console biasa. Untuk contoh kali ini, pilihlah *KDE2 Mini* dan selanjutnya klik tombol *Next*.



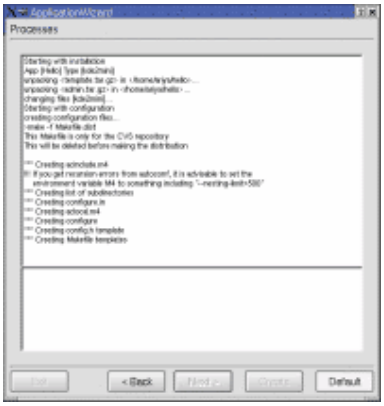
Gambar 3. Application Wizard

Selanjutnya Anda harus mengatur beberapa setting umum seperti nama aplikasi, direktori kerja, nama pembuat, dan lain sebagainya (Gambar 4). Anda bisa mengikuti Gambar 4 dengan mengisikan *Hello* sebagai nama aplikasi. Sementara itu direktori kerjanya dapat Anda tentukan sendiri, lantas klik tombol *Next*.



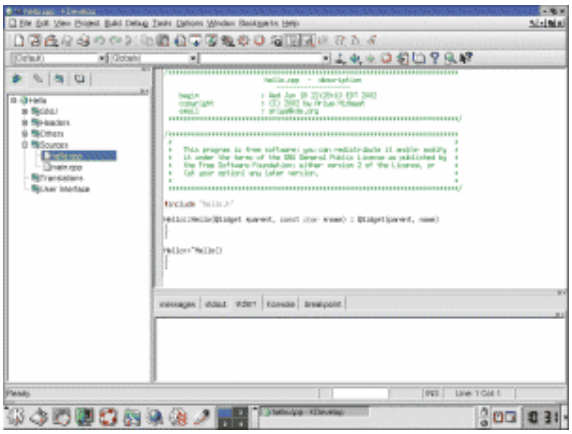
Gambar 4. Menentukan setting project

Dalam beberapa saat, wizard-nya KDevelop ini akan bekerja menyusun file-file yang diperlukan, termasuk juga file-file automake dan autoconf, script konfigurasi, struktur manual dan dokumentasi API, serta hal-hal kecil lainnya. Perbedaan KDevelop dengan Microsoft Visual Studio akan terlihat di sini, KDevelop menghasilkan dan mengolah file-file yang diperlukan berdasarkan standar pengembangan aplikasi Unix (seperti automake dan autoconf). Setelah pekerjaan kecil ini selesai, tinggal klik tombol *Create* dan wizard pun selesai.



Gambar 5. Menyusun file-file kerja

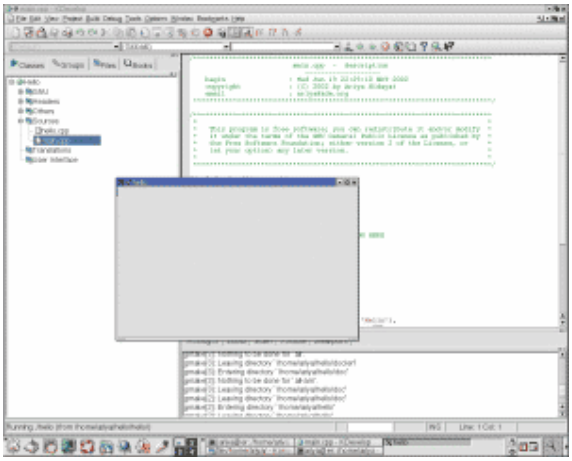
Sekarang KDevelop sudah mengaktifkan project baru yang bernama *Hello* tadi sehingga tampilannya berubah menjadi seperti pada Gambar 6. Pada panel bagian kiri bisa Anda cermati bahwa KDevelop sudah membuatkan dua file source-code yang diperlukan, yaitu *main.cpp* dan *hello.cpp*. Sementara itu panel sebelah kiri menunjukkan editor yang sedang memuat file *hello.cpp*.



Gambar 6. Mengedit Hello di KDevelop

Percaya atau tidak, dua file yang telah dengan baik hati dibuatkan oleh KDevelop ini sudah memenuhi syarat sebagai aplikasi KDE. Untuk membuktikannya, Anda bisa melakukan compile sekaligus menjalankannya dengan memilih menu *Build*, *Execute* (shortcutnya adalah F9). Atau bisa juga dengan tombol *Run* di toolbar, yakni yang iconnya bergambar roda gigi. Sementara proses compile dilakukan, perkembangan dan statusnya bisa Anda lihat di panel kanan bawah pada tab *Messages*.

Bila tidak ada yang salah, artinya belum ada yang Anda utak-atik, mestinya program Hello ini sudah tampak. Karena masih sederhana sekali, program Hello ini hanya menampilkan window kosong (Gambar 7).



Gambar 7. Program Hello

Untuk menghasilkan sebuah aplikasi yang betul-betul punya fungsi, tidak hanya sekedar menayangkan window kosong, Anda dapat memodifikasi file source code *main.cpp* dan *hello.cpp*. Karena KDE dan Qt dibangun dengan bahasa C++, maka mau tidak mau Anda harus terlebih dahulu menguasai bahasa C++ ini. Di samping itu, untuk merancang aplikasi yang menggunakan pustaka yang telah disediakan oleh Qt dan KDE, maka prasyarat lainnya adalah sudah mengenali pustaka-pustaka tersebut. Sebagai langkah awal, Anda bisa saja mempelajari Qt terlebih dahulu. Tutorial berseri tentang pengenalan pemrograman Qt sempat dimuat di InfoLINUX, yaitu dari no 7/2001.

Guna bereksperimen sambil berkenalan dengan kelas-kelas C++ di Qt dan KDE, Anda bisa memodifikasi program Hello sesuai panduan berikut ini. Mula-mula definisi kelas *Hello* yang ada di file header *hello.h* mesti diubah. Asalnya kelas Hello ini diturunkan dari kelas QWidget, yaitu kelas dasar untuk widget pada Qt. Sayangnya, kelas dasar QWidget ini tidak banyak berguna. Karenanya lebih baik menggantinya dengan kelas KMainWindow yang merupakan pembentuk window utama di sebuah aplikasi KDE.

Menyunting file *hello.h* bisa dilakukan dengan memilihnya dari daftar file (klik tab *Files* di panel kiri). Editlah file tersebut sehingga menjadi seperti di bawah ini. Perhatikan bahwa komentar pada bagian awal file (yang dihasilkan KDevelop) tidak dikutip di sini, Anda bisa membiarkan sebagaimana adanya.

```
#ifndef HELLO_H
#define HELLO_H

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <kapp.h>
#include <qwidget.h>

#include <kmenubar.h>
#include <kmainwindow.h>

/** Hello is the base class of the project */
```

```
class Hello : public KMainWindow
{
  Q_OBJECT
  public:
    /** construtor */
    Hello(QWidget* parent=0, const char *name=0);
    /** destructor */
    ~Hello();

  protected:
    void initGUI();
    void updateGUI();
    void load( const char *fname );
    void save( const char *fname );

  private slots:

    void fileOpen();
    void fileSave();

  private:

    QString filename;
};

#endif
```

Dari kelas Hello yang baru ini dapat dicermati bahwa beberapa fungsi telah dibubuhkan, yaitu *initGUI()* untuk inisialisasi antarmuka, *updateGUI()* untuk melakukan update tampilan, serta dua fungsi *load()* dan *save()* yang mencontohkan cara memuat dan menyimpan file. Sebagai tambahan, fungsi *fileOpen()* dan *fileSave*() merupakan slot untuk menerima signal dari menu. Untuk lebih jelasnya mengenai signal dan slot, Anda bisa merujuk ke tutorial pemrograman Qt di InfoLINUX no 7/2001 atau bisa juga di dokumentasi Qt.

Selanjutnya, bagian utamanya yakni file *hello.cpp* juga diubah menjadi seperti di bawah ini:

```
#include "hello.h"

#include <kfiledialog.h>
#include <ktoolbar.h>
#include <kstatusbar.h>
#include <kiconloader.h>
#include <kaction.h>
#include <kstdaction.h>
#include <kmainwindow.h>
#include <kpopupmenu.h>
#include <kurl.h>

QString filter = "*.*|All";

Hello::Hello(QWidget *parent, const char *name) :
  KMainWindow( 0, "Hello", WDestructiveClose )
{
  initGUI();
```

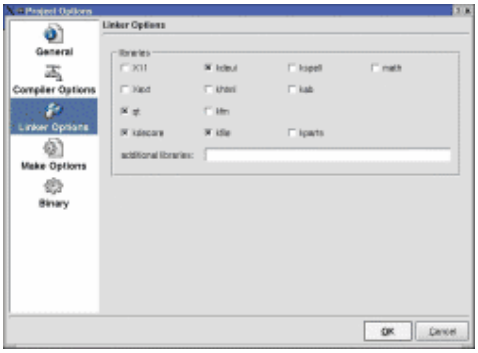
```
  updateGUI();
  show();
}

Hello::~Hello()
{
}

// initializes user-interface
void Hello::initGUI()
{
  KStdAction::open(this, SLOT( fileOpen() ),actionCollection(), "open" );
  KStdAction::save(this, SLOT( fileSave() ),actionCollection(), "save" );
  KStdAction::close(this, SLOT( close() ),actionCollection(), "close" );

  actionCollection()->action("open")->plug( toolBar() );
  actionCollection()->action("save")->plug( toolBar() );

  KPopupMenu * file = new KPopupMenu( this );
  menuBar()->insertItem( "&File", file );
  actionCollection()->action("open")->plug( file );
  actionCollection()->action("save")->plug( file );
  actionCollection()->action("close")->plug( file );

  menuBar()->insertItem( "&Help", helpMenu() );
}

// updates user-interface
void Hello::updateGUI()
{
  setCaption( filename );
}

// loads from file
void Hello::load( const char *fname )
{
  // do file loading here

  filename = fname;

  // update caption and titlebar
  statusBar()->message( filename + " loaded", 2000 );
  updateGUI();
}

// saves to file
void Hello::save( const char *fname )
{
  // do file saving here

  filename = fname;

  // update caption and titlebar
  statusBar()->message( filename + " saved", 2000 );
  updateGUI();
}

// opens and loads from file
void Hello::fileOpen()
```

```
{
  KURL url;
  url = KFileDialog::getOpenURL( QString::null, filter, this );

  if( url.isEmpty() ) return;
  if( !url.isLocalFile() ) return;

  load( url.fileName() );
}

// saves to file
void Hello::fileSave()
{
  KURL url;
  url = KFileDialog::getSaveURL( QString::null, filter, this );

  if( url.isEmpty() ) return;
  if ( !url.isLocalFile() ) return;

  save( url.fileName() );
}
```

Konstruktor kelas Hello cukup sederhana, yaitu hanya menginisialisasi tampilan dan mengupdatenya untuk pertama kali. Sementara itu yang dikerjakan fungsi *initGUI()* adalah membuat tiga *action*, masing-masing untuk *fileOpen(), fileSave(),* dan c*lose()*. Melalui kelas *KStdAction*, ketiga action dapat dipasangkan (*plug*) ke menu dan toolbar dengan mudah. Di sisi lain, fungsi *updateGUI()* hanya akan memperbaharui *caption* dari program jika *filename* berubah.

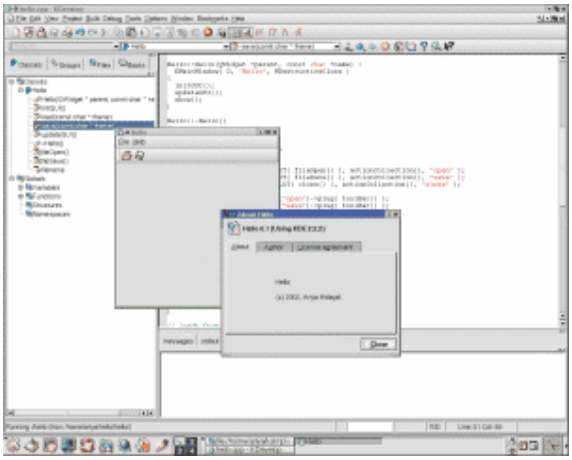
Baik fungsi *load()* dan *save()* pada program Hello di atas tidak melakukan operasi apapun karena hanya berfungsi sebagai contoh. Kedua slot *fileOpen()* dan *fileSave*() akan dipanggil dari *action* yang bersesuaian, lihat juga fungsi *initGUI()*. Pada badan fungsi masing-masing hanya ada pemanggilan terhadap *KFileDialog* yang akan menghasilkan kotak dialog untuk *Load* atau *Save*. Bisa dilihat, secara langkah demi langkah, memang inilah yang akan terjadi pada aplikasi yang sungguhan.

Hingga saat ini, kalau Anda langsung melakukan compile kemungkinan Anda akan menjumpai kesalahan karena pustaka yang bernama *kfile* belum di-link dengan program Hello. Untuk mengatasinya, Anda harus mengubah pilihan *linker* yang digunakan. Pilih menu *Project*, *Options* atau dengan shortcut F7 sehingga tampil kotak dialog *Project Options*. Klik pada item *Linker Options* di daftar sebelah kiri dan selanjutnya tandai checkbox yang bertuliskan *kfile*. Sekarang Anda bisa mengkompilasi program Hello ini tanpa masalah.



Gambar 8. Mengutak-atik Project Options

Kalau tidak terjadi salah ketik ketika menyunting source-code, mestinya program Hello sudah berubah menjadi seperti Gambar 9. Terlihat bahwa baik menu dan toolbar sudah ada pada window program, bahkan kotak dialog *Open* atau *Save* sudah bisa muncul jika tombol pada toolbar diklik.

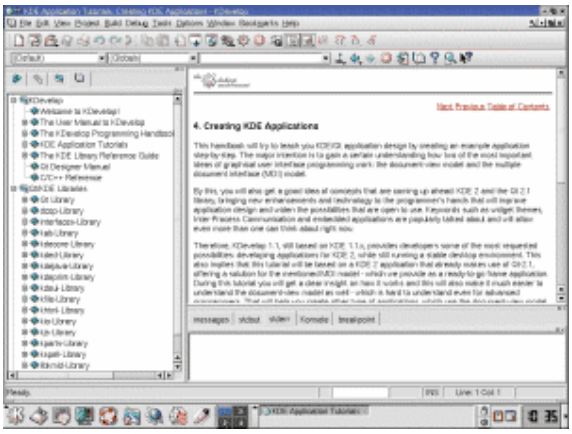


Gambar 9. Program Hello yang baru

Mendistribusikan aplikasi yang Anda buat juga tidak kalah gampang, tinggal pilih menu *Project, Make distribution*. (yang sudah mengerti banyak tentang Makefile pastilah sudah hafal cara lainnya: *make dist*). Selang beberapa saat file *hello-0.1.tar.gz* akan terbentuk dan inilah tarbal dari source-code program Hello. Angka 0.1 di situ menunjukkan versi 0,1 (ingatlah langkah ke-2 dari Application Wizard). Anda sekarang bisa menyebarkan tarbal ini, misalnya ke teman-teman yang juga menggunakan Linux. Bila ada yang tertarik, mereka bisa mengambil tarbal tersebut, mengekstraknya dengan perintah *tar zxvf hello-0.1.tar.gz*, menjalankan *./configure* yang diikuti dengan *make*. Tentu saja, kalau program Anda cuman sekedar menampilkan "Hello, world!" mungkin tidak banyak yang akan terpikat mencobanya.

Kadang-kadang, meskipun telah mengenal sebagian pustaka Qt dan KDE yang paling penting, keberadaan dokumentasi masih mutlak diperlukan. Tetapi jangan khawatir karena KDevelop menyediakan fasilitas untuk mengakses dokumentasi ini dengan cepat,

bahkan tanpa berpindah dari KDevelop. Cukup aktifkan tab Dokumentasi (yang iconnya bergambar buku terbuka) pada panel kiri sehingga akan didaftar berbagai macam dokumentasi yang sudah diinstal di sistem Anda. Dari sini tinggal klik mana yang Anda perlukan dan dalam sekejap panel sebelah kanan akan menghadirkan versi HTML dari dokumentasi tersebut (Gambar 10), tinggal disimak dan dipahami baik-baik dan satu saat Anda telah siap menjadi developer KDE. Selamat belajar !



Gambar 10. Mengakses dokumentasi Qt/KDE

Menggunakan Automake dan Autoconf

Tulisan ini berisi informasi tentang cara menggunakan automake dan autoconf. Kedua utility ini dipergunakan guna menghasilkan suatu sistem pengkonfigurasi dan instalasi aplikasi secara otomatis.

Untuk mempelajari automake dan autoconf, Anda tentunya harus sudah akrab dengan metode pengembangan aplikasi menggunakan GNU C Compiler, yaitu yang berkaitan dengan `gcc`, `make`, dan utility-utility lainnya. Hal ini diperlukan karena pada dasarnya automake dan autoconf merupakan pelengkap bagi sistem pengembangan aplikasi ala GNU.

Sekilas tentang automake dan autoconf

Rata-rata sebagian besar aplikasi Unix memiliki metoda compile dan instalasi yang sama. Sebagai contoh, katakanlah terdapat sebuah utility yang dikemas dalam file distribusi `contoh-0.1.tar.gz`. Cara melakukan compile dan instalasi adalah cukup dengan perintah-perintah berikut ini:

```
tar zxvf contoh-0.1.tar.gz
cd contoh-0.1
./configure
make
make install
```

Perintah `configure` akan menjalankan sebuah script dan akan melakukan serangkaian proses pendeteksian sistem Unix yang digunakan untuk mendapatkan informasi tertentu. Dari hasil `configure`, maka akan terbentuk `Makefile` yang disusun berdasarkan template yang tersimpan di `Makefile.in`. Ketika Anda menjalankan make, maka proses compile akan dimulai dan biasanya hal ini adalah untuk menghasilkan program utama. Selanjutnya, untuk menginstal program `contoh-0.1` ini, sekali lagi `make` harus dijalankan dengan target bernama '`install`'. Instalasi lumrahnya hanyalah proses penyalinan file hasil compile ke direktori-direktori yang sesuai, program ke `/usr/local/bin`, pustaka fungsi (library) ke `/usr/local/lib`, file-file header ke `/usr/local/include`, serta panduan penggunaan ke ke `/usr/local/man` atau `/usr/local/info`.

Terkadang, instalasi dapat saja tidak dilakukan ke `/usr/local/`. Sebuah program yang ingin digunakan secara pribadi oleh user yang menginstal saja, dapat diinstal ke direktori homenya, misalnya `/home/ariya/`. Dalam hal ini, satu-satunya yang dimodifikasi adalah proses `configure` yang harus mengikutkan option untuk *prefix*, sebagaimana ditunjukkan contoh berikut:

```
./configure --prefix=/home/ariya
```

Dapat dilihat bahwa proses compile dan instalasi sebuah program dari source codenya relatif cukup mudah untuk dilakukan oleh user. Hal ini tidak lain karena proses konfigurasi sudah dilakukan secara otomatis oleh si `configure`.

Misalkan Anda mempunya aplikasi yang Anda kembangkan sendiri, maka proses compile dan instalasi aplikasi tersebut juga dapat memanfaatkan konfigurasi otomatis dengan script `configure`. Hal ini dapat dilakukan dengan memanfaatkan automake dan autoconf.

Automake adalah utility untuk mempermudah membuat Makefile. Bagi Anda yang pernah menangani aplikasi yang cukup besar dan mencakup beberapa puluh (atau bahkan ratusan) file, membuat dan mengelola Makefile merupakan pekerjaan yang menyita waktu dan energi. Dengan automake, cukup didefinisikan beberapa hal yang perlu saja maka akan dihasilkan sebuah Makefile yang lengkap untuk proses compile aplikasi tersebut. Makefile yang dibuat juga bahkan telah dibekali kemampuan untuk menghasilkan file .tar.gz, yang merupakan distribusi paket dari aplikasi yang dikembangkan.

Autoconf adalah sekumpulan utility yang digunakan untuk memudahkan proses konfigurasi program pada aneka varian sistem. Tanpa menggunakan konfigurasi yang sesuai, maka program yang dirancang berjalan di berbagai platform harus melakukan pendeteksian ragam sistem yang digunakan untuk menyesuaikan hal-hal khusus, misalnya pendeteksian library yang dibutuhkan, panggilan fungsi ke pustaka tertentu, inkompatibilitas antar fungsi, dan lain sebagainya. Autoconf dikemas dalam beberapa program, mencakup `autoconf`, `autoheader`, `autoscan`, `autoreconf`, `autoupdate`, dan `ifnames`.

Untuk programer sendiri, automake dan autoconf akan menguntungkan karena beberapa hal, yaitu:

- Tidak perlu membuat Makefile secara manual. Bagi yang pernah terlibat pengembangan aplikasi yang kompleks, dengan sekian belas sub-direktori, maka menghasilkan Makefile yang tepat akan menyita banyak tenaga (belum lagi menentukan dependencies). Automake akan membantu tahapan ini.
- Makefile yang dihasilkan automake sudah mempunyai target bernama `install` dan `uninstall`. Keduanya akan digunakan oleh user untuk menginstal ataupun menghapus instalasi program. Hal ini berarti programer tidak perlu lagi mengurusi soal instalasi ini.
- Makefile yang dihasilkan automake juga mempunyai target bernama `dist`. Target ini digunakan untuk menghasilkan sebuah package dalam bentuk file `tar.gz` yang dapat langsung didistribusikan.
- Untuk aplikasi yang dirancang portable di beberapa sistem Unix yang berbeda (misalnya AIX, Solaris, dan Linux), hasil konfigurasi dari `configure` akan berguna untuk mereduksi sejumlah `#ifdef` yang lazimnya dimanfaatkan untuk mendeteksi dan mengatasi perbedaan-perbedaaan di antara sistem Unix tersebut.

Instalasi

Sebelum dapat menggunakan automake dan autoconf, sistem Anda tentunya harus dilengkapi dengan utility yang sesuai. Instalasi bisa dilakukan dari paket `autotools`. Jika Anda menggunakan Red Hat Linux 6.1, maka cukup instal beberapa file RPM berikut ini:

```
autoconf-2.13-5.noarch.rpm
automake-1.4-5.noarch.rpm
```

```
libtool-1.3.3-1.noarch.rpm
```

Tentu saja, perangkat pengembangan program GNU C Compiler juga harus terpasang terlebih dahulu. Hal ini mencakup compiler `gcc` (atau `egcs` atau `pgcc`) beserta kelengkapannya. Jangan lupa untuk mengikutsertakan `m4` di package `m4-1.4-12.i386.rpm`. Ini adalah macro processor yang dibutuhkan oleh autoconf.

Bila Anda ingin melakukan sendiri proses compile, Anda harus mengambil source codenya dari file `tar.gz` yang sesuai, kemudian melakukan `configure`, `make`, dan `make install`. Sudah barang tentu paket autotools ini juga dibangun menggunakan autoconf dan automake.

Source code beserta program lengkap plus dokumentasi autoconf dan automake dapat diperoleh di situs web GNU (www.gnu.org).

Memodifikasi Aplikasi Lama

Aplikasi yang sudah ada dapat dengan mudah dimodifikasi agar memanfaatkan autoconf. Syarat untuk ini adalah bahwa aplikasi Anda sudah memiliki Makefile yang digunakan untuk melakukan `make`.

Berikut adalah langkah-langkah yang perlu dilakukan:

1) Paksa agar terdapat script untuk `configure`. Caranya adalah dengan perintah-perintah berikut ini. Lakukan pada direktori yang menyimpan source-code dari aplikasi tersebut:

```
cp Makefile Makefile.orig
mv Makefile Makefile.in
autoscan
mv configure.scan configure.in
autoheader
autoconf
```

Catatan: Bila terdapat sub-direktori, maka yang harus dilakukan di masing-masing sub direktori agak berbeda, yaitu cukup:

```
cp Makefile Makefile.orig
mv Makefile Makefile.in
autoheader
```

2) Modifikasi file `configure.in`. Hal ini dilakukan agar mudah melakukan rekonfigurasi bila file *.in harus diubah lagi. Pada baris terakhir terdapat

```
AC_OUTPUT(Makefile)
```

Ini harus diubah menjadi:

```
AC_OUTPUT(Makefile, echo timestamp > stamp-h)
```

3) Nah, kini hasil autoconf siap untuk dicoba. Silakan jalankan:

```
./configure
```

Selamat ! Kini aplikasi Anda telah dapat dikonfigurasi otomatis. Namun demikian, karena dalam kasus ini Makefile awalnya sudah tersedia, tidak dihasilkan dari automake, maka belum tentu keseluruhan program dapat dipaketkan menjadi file distribusi dan diinstal oleh user dengan mudah. Hal ini sangat tergantung bagaimana isi dari Makefile tersebut.

Membuat Aplikasi Baru

Membuat aplikasi baru agar menggunakan automake dan autoconf jauh lebih mudah dibanding melakukan konversi aplikasi lama. Namun demikian, agar proses ini menjadi mudah dipahami, berikut adalah hal-hal yang patut dijadikan pegangan:

- Script `configure` dihasilkan autoconf dari file `configure.in`
- Pada saat dieksekusi, script `configure` menghasilkan `Makefile` dari file `Makefile.in`
- File `Makefile.in` dihasilkan menggunakan automake, untuk ini dibutuhkan `Makefile.am`

Dapat dilihat bahwa yang perlu disediakan oleh sang programer hanyalah file `Makefile.am` dan `configure.in` (serta source code program tentunya!).

Berikut adalah panduan langkah demi langkah untuk memulai aplikasi baru dengan automake dan autoconf.

Mula-mula tentunya kita harus menulis source code untuk aplikasi baru tersebut. Karena melibatkan sejumlah file, ada baiknya aplikasi ini diletakkan di direktori khusus. Berikut adalah contohnya:

```
$ mkdir hello
$ cd hello
$ cat > hello.c
#include <stdio.h>
int main(int argc, char*argv[])
{
  printf ("Hello, World ! (in autoconf)\n");
}
<Ctrl+D>
$
```

Contoh di atas hanya sekedar contoh. Tentunya Anda ingin membuat aplikasi yang lebih bermanfaat dari sekedar "Hello, World!".

Secara manual, program di atas dapat dikompilasi dan dijalankan dengan cara:

```
$ gcc -o hello hello.c
$ ./hello
```

Dan akan muncul pesan (yang sangat terkenal) berikut ini:

```
Hello, World ! (in autoconf)
```

Sekarang, kita akan melakukan proses compile via automake dan autoconf. Untuk ini diperlukan file `Makefile.am` dan `configure.in`.

File `Makefile.am` adalah sebagai berikut:

```
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```

Sementara file `configure.in` adalah:

```
AC_INIT(hello.c)
AM_INIT_AUTOMAKE(hello,1.0)
AC_PROG_CC
AC_PROG_INSTALL
AC_OUTPUT(Makefile)
```

Penting untuk diingat bahwa file `Makefile.am` hanya akan berisi *assignment*, yaitu hanya relasi-relasi saja dan bukan instruksi atau perintah untuk dijalankan (lazim juga dikenal sebagai *logic language*). Sebaliknya, file `configure.in` berisi daftar instruksi yang harus dieksekusi (*procedural language*).

Berikut adalah penjelasan ringkas dari hal-hal yang dikerjakan sebagai instruksi untuk contoh file `configure.in` di atas:

- Instruksi `AC_INIT` dilakukan untuk menginisialisasi script konfigurasi. Untuk perintah ini, diperlukan nama file utama yang merupakan bagian dari source code.
- Instruksi `AM_INIT_AUTOMAKE` diperlukan karena di sini Makefile-nya dihasilkan dengan automake. Bila `Makefile.in` dibuat manual, hal ini tidak perlu dilakukan. Yang perlu diperhatikan di sini adalah bahwa argument pertama adalah nama paket programnya (yaitu *hello*) sedangkan argument keduanya adalah nomor versi (dalam hal *1.0*).
- Instruksi `AC_PROG_CC` memeriksa jenis compiler C yang digunakan.
- Instruksi `AC_PROG_INSTALL` mendeteksi kehadiran utility instal ala BSD. Jika hal ini tidak tersedia, maka akan digunakan script `install-sh` sebagai gantinya.
- Instruksi `AC_OUTPUT` akan menghasilkan `Makefile` dan `Makefile.in`.

Karena standar GNU mensyaratkan beberapa file pelengkap, maka kita dapat membuat file-file ini dengan cepat menggunakan perintah `touch`. Pada kenyataannya, mungkin Anda sudah mempunyai file-file ini. Apabila tidak, maka gunakan perintah seperti berikut:

```
$ touch NEWS README AUTHORS ChangeLog
```

Bila sebelumnya file-file ini tidak ada, maka kini file tersebut sudah muncul sendiri dan mungkin kelak dapat Anda sunting agar berisi informasi yang (lebih) bermanfaat.

Yang tidak kalah pentingnya diperlukan adalah script `install-sh`. Tanpa script ini, automake akan menolak untuk dijalankan. Script `install-sh` ini sendiri dapat didapat dari berbagai program yang menggunakan automake dan autoconf (termasuk automake

dan autoconf sendiri!). Jika sistem Anda memiliki automake dan autoconf, maka script ini biasanya bisa dijumpai di direktori `/usr/share/automake`. Silakan copy file ini ke direktori yang sekarang aktif sehingga juga menjadi bagian dari program *hello*.

Nah, sekarang tiba bagian yang seru. Jalankan autoconf dengan perintah-perintah:

```
$ aclocal
$ autoconf
```

Apa yang sebenarnya terjadi ? Perintah `aclocal` menyiapkan file `aclocal.m4`. Secara mudahnya, file ini diperlukan gara-gara adanya perintah `AM_INIT_AUTOMAKE` pada `configure.in`. Selanjutnya perintah `autoconf` menggabungkan isi dari `aclocal.m4` dan `configure.in` serta menghasilkan script `configure`.

Tahap berikutnya adalah menjalankan automake:

```
[ariya@labkon hello]$ automake -a
automake: configure.in: installing `./mkinstalldirs'
automake: configure.in: installing `./missing'
automake: Makefile.am: installing `./INSTALL'
automake: Makefile.am: installing `./COPYING'
```

Pesan-pesan yang tampil adalah karena beberapa file-file tertentu tidak ada. Untungnya, automake berbaik hati membuat file-file tersebut secara otomatis (karena digunakan option -a).

Sekarang aplikasi *hello* ini sudah siap.

Untuk melakukan compile, tinggal dilakukan:

```
$ ./configure
creating cache ./config.cache
checking for a BSD compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking whether make sets ${MAKE}... yes
checking for working aclocal... found
checking for working autoconf... found
checking for working automake... found
checking for working autoheader... found
checking for working makeinfo... found
checking for gcc... gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking for a BSD compatible install... /usr/bin/install -c
updating cache ./config.cache
creating ./config.status
creating Makefile
$ ./make
gcc -DPACKAGE=\"hello\" -DVERSION=\"1.0\"  -I. -I.      -g -O2 -c hello.c
gcc  -g -O2  -o hello  hello.o
```

Yang dilakukan `configure` adalah mendeteksi platform sistem yang aktif, kemudian menghasilkan Makefile yang tepat sebagaimana telah ditentukan dalam file `Makefile.in`.

Lebih jauh, program *hello* ini bisa diinstal ke `/usr/local/`. Cukup dengan perintah:

```
# make install
make[1]: Entering directory `/home/ariya/hello'
/bin/sh ./mkinstalldirs /usr/local/bin
  /usr/bin/install -c  hello /usr/local/bin/hello
make[1]: Nothing to be done for `install-data-am'.
make[1]: Leaving directory `/home/ariya/hello'
```

(Dalam kebanyakan kasus, Anda harus login dahulu sebagai *root* agar bisa melakukan install).

Hasil instalasi bisa diperiksa dengan:

```
$ /usr/local/bin/hello
Hello, World ! (in autoconf)
```

Membatalkan instalasi juga sama mudahnya. Cukup dengan:

```
# make uninstall
list='hello'; for p in $list; do \
rm -f /usr/local/bin/`echo $p|sed 's/$//'|sed 's,x,x,'|sed 's/$//'`; \
done
```

Maka file `/usr/local/bin/hello` akan hilang dalam sekejap.

Salah satu feature yang amat berguna adalah membuat file distribusi, yaitu source code yang dikemas dalam file tar.gz (atau semacamnya). Hal ini bisa dilakukan dengan:

```
$ make dist
rm -rf hello-1.0
mkdir hello-1.0
chmod 777 hello-1.0
here=`cd . && pwd`; \
top_distdir=`cd hello-1.0 && pwd`; \
distdir=`cd hello-1.0 && pwd`; \
cd . \
&& automake --include-deps --build-dir=$here --srcdir-name=.
--output-dir=$top_distdir --gnu Makefile
chmod -R a+r hello-1.0
GZIP=--best gtar chozf hello-1.0.tar.gz hello-1.0
rm -rf hello-1.0
```

Proses ini akan menghasilkan file `hello-1.0.tar.gz`. Anda dapat memberikan file ini kepada orang lain dan menyebutnya sebagai distribusi program *hello*. Yang tertarik dapat memanfaatkan dan melakukan kompilasi program *hello*, cukup dengan perintah-perintah `tar zxvf`, `configure`, dan `make` (sebagai sudah disebutkan di awal).

Menangani Aplikasi yang Kompleks

Langkah-langkah di atas sudah cukup memadai untuk pengembangan aplikasi yang sederhana. Sekarang misalnya Anda mempunyai program yang source code tersebar di beberapa direktori, menggunakan pustaka fungsi (library), campuran program C dan C++, ataupun menggunakan lex (atau flex) dan yacc (atau bison), harus dilakukan langkah-langkah tambahan.

Bila program Anda juga harus dicompile dengan C++ (atau berisi campuran file C dan CPP), maka yang harus ada pada configure.in adalah:

```
AC_PROG_CPP
```

Bila aplikasi yang akan dibangun membutuhkan library tertentu, hal ini dapat dispesifikasi dengan relasi `LDADD` pada Makefile.am, contohnya adalah:

```
hello_LDADD = -lX11
```

Untuk menangani beberapa direktori, caranya cukup sederhana. Cukup ubah sedikit instruksi `AC_OUTPUT` pada `configure.in`. Misalnya, kini terdapat sub direktori `src/`, maka file `configure.in` harus menjadi:

```
AC_OUTPUT(Makefile src/Makefile)
```

Juga jangan lupa untuk membubuhkan relasi `SUBDIRS` pada `Makefile.am`, sebagaimana contoh berikut:

```
SUBDIRS = src
```

Kemudian, pada sub direktori `src/` ini juga harus terdapat `Makefile.am` yang tepat, misalnya:

```
bin_PROGRAMS = hello
hello_SOURCES = main.c
```

Kelak pada saat automake dan autoconf dijalankan, sub direktori `src/` akan ikut ditelusuri sehingga dapat dihasilkan `Makefile.in` dan `Makefile` yang sesuai.

Bila aplikasi Anda menggunakan *lexical analyzer* dan *semantic parser* yang dikonstruksi menggunakan `flex` dan `bison` (atau lex dan yacc), maka file `configure.in` perlu disesuaikan dengan membubuhkan baris-baris berikut:

```
AM_PROG_LEX
AC_PROG_YACC
```

File input untuk `flex` dan `bison` tinggal dimasukkan sebagai bagian dari SOURCES untuk aplikasi atau library (pada file `Makefile.am`), misalnya:

```
bin_PROGRAMS = hello
hello_SOURCES = main.c grammar.y parser.l
```

Catatan: automake mengasumsikan file `.c` yang dihasilkan dari `.l` dan `.y` adalah sama dengan nama file sumbernya tetapi dengan mengganti suffiksnya (misalnya `bslex.l` akan menghasilkan `bslex.c`).

Bila program Anda menghasilkan sebuah pustaka fungsi (library), maka `configure` harus mampu menangani `ranlib`. Oleh karenanya, pertama-tama, jangan lupa tambahan instruksi `AC_PROG_RANLIB` pada configure.in. Adapun relasi-relasi yang dibutuhkan di `Makefile.am` untuk membuat sebuah library dicontohkan berikut ini:

```
lib_LIBRARIES = libfoo.a
libfoo_a_SOURCES = foo.c private.h
libfoo_a_LIBADD = objfile.o
libfoo_a_DEPENDENCIES = dep1 dep2
```

Berikut adalah penjelasan masing-masing relasi di atas:

- `lib_LIBRARIES` adalah mirip seperti `bin_PROGRAMS` tetapi kali ini berlaku untuk pustaka fungsi(library). Di sini dapat didaftar library-library yang kelak akan diinstal ke `/usr/local/lib`
- `libfoo_a_SOURCES` menentukan file-file source code untuk library bernama `libfoo.a`. Dalam hal ini dapat juga dimasukkan file-file header yang merupakan bagian dari source code, tapi tidak akan didistribusikan bersama `libfoo.a`.
- `libfoo_a_DEPENDENCITES` menentukan target lain yang harus tersedia terlebih dahulu sebelum library ini dapat dihasilkan
- `libfoo_a_LIBADD` adalah untuk mendaftar file-file objek yang akan dianggap sebagai bagian dari library ini.

Bila aplikasi Anda perlu menghasilkan library untuk program utama namun library tersebut bukan bagian dari distribusi (tidak akan diinstal ke `/usr/local/lib`), maka dapat digunakan relasi `noinst_LIBRARIES` pada `Makefile.am` seperti dicontohkan di bawah ini:

```
noinst_LIBRARIES = libdummy.a
libdummy_a_SOURCES = dummy.h dummy.c
```

Mengatasi Variansi Sistem

Telah disebut-sebut bahwa script `configure` akan mendeteksi jenis sistem dan platform Unix. Hasilnya dapat digunakan agar program aplikasi untuk mengetahui dan menangani variansi maupun ketidakkompatibelan antar sistem. Sebagai contoh, lazim dijumpai bahwa program harus dapat bekerja pada sistem yang low-endian dan big-endian. Autoconf dapat diinstruksikan untuk mendeteksi endianness dari sistem ini dengan `AC_C_BIGENDIAN`. Bila ternyata sistem memang big endian, maka akan terdefinisi `WORDS_BIGENDIAN`. Sebagai contoh, asumsikan terdapat program `test.c` sebagaimana berikut ini:

```
#include <stdio.h>
int main(int argc, char*argv[])
{
  printf ("Hello, World !\n");
```

```
#ifdef WORDS_BIGENDIAN
  printf("Big Endian\n");
#endif
}
```

dengan instruksi `AC_C_BIGENDIAN` yang disisipkan pada `configure.in` (harus setelah `AC_INIT`).

Setelah proses automake dan autoconf, akan dihasilkan script `configure` yang sudah mampu mendeteksi endianness dari sistem. Ketika dijalankan, maka akan diperoleh:

```
creating cache ./config.cache
checking for a BSD compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking whether make sets ${MAKE}... yes
checking for working aclocal... found
checking for working autoconf... found
checking for working automake... found
checking for working autoheader... found
checking for working makeinfo... found
checking for gcc... gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking whether byte ordering is bigendian... yes
checking for a BSD compatible install... /usr/bin/install -c
updating cache ./config.cache
creating ./config.status
creating Makefile
```

Perhatikan bahwa sekarang terdapat baris yang menyatakan *checking whether byte ordering is bigendian...*.

Nah, sekarang apabila program *test* ini dicompile dan dijalankan, akan diperoleh hasil:

```
Hello, World !
Big Endian
```

bila memang proses konfigurasi, compile, dan instalasi dilakukan pada sistem yang big endian (misalnya HP-UX dan SPARC). Akan tetapi, bila program yang sama dikonfigurasi, dicompile, dan diinstal pada Linux untuk Intel Pentium, hasil yang diperoleh adalah:

```
Hello, World !
```

Dari ilustrasi ini dapat disimpulkan bahwa penggunaan instruksi-instruksi tertentu dengan autoconf akan memudahkan program yang dikembangkan untuk menyesuaikan diri dengan lingkungan (baik sistem operasi ataupun platform).

Disamping `AC_C_BIGENDIAN`, terdapat pula instruksi-instruksi lain yang cukup berguna. Beberapa di antaranya disajikan di bawah ini (daftar lengkapnya sendiri dapat dilihat di manual penggunaan autoconf).

`AC_C_CHAR_UNSIGNED`
: Bila tipe `char` adalah unsigned, maka akan didefinisikan `__CHAR_UNSIGNED__`.

`AC_CHECK_SIZEOF (type)`
: Ukuran dari tipe yang disebutkan (*type*) akan diletakkan pada definisi `SIZEOF_xtype`. Ukuran ini dinyatakan dalam byte. `xtype` sendiri dibentuk dari *type* dengan mengubah spasi menjadi garis bawah dan semuanya dalam huruf kecil. Untuk pointer, maka terdapat prefix P. Sebagai contoh, pada HP-UX 10.x, `AC_CHECK_SIZEOF(double)` akan menghasilkan `SIZEOF_DOUBLE` sebesar 8 dan `AC_CHECK_SIZEOF(char*)` akan menghasilkan `SIZEOF_CHAR_P` sebesar 4.

`AC_TYPE_SIZE_T`
: Bila `size_t` tidak didefinisikan, maka `size_t` akan dianggap sebagai unsigned.

`AC_CHECK_HEADERS(headerfile)`
: Periksa *headerfile*, bila ada, maka akan didefinisikan `HAVE_headerfile`. Sebagai contoh, `AC_CHECK_HEADERS(unistd.h)` akan menghasilkan `HAVE_UNISTD_H` jika memang file `unistd.h` dijumpai pada sistem.

`AC_CHECK_FUNCS(func)`
: Periksa apakah sistem mempunyai fungsi *func*. Bila ya, maka akan didefinisikan `HAVE_func`. Sebagai contoh, `AC_CHECK_FUNCS(memcpy)` akan menghasilkan `HAVE_MEMCPY` pada sistem yang mendukung ANSI C.

`AC_CHECK_LIB(lib,func)`
: Periksa apakah terdapat fungsi bernama *func* pada library *lib*. Bila ada, maka akan didefinisikan `HAVE_lib`. Contohnya adalah `AC_CHECK_LIB(X11,XGetImage)` akan menghasilkan `HAVE_X11` pada semua sistem yang mempunyai X Window.

Sebagai catatan akhir, dapat dilihat bahwa metoda di atas bekerja dengan memanfaatkan option -D dari compiler (misalnya `gcc`). Hal ini akan jelas terlihat kalau Anda mengamati output dari `make`.

Cara lain untuk mendapatkan definisi-definisi selain option -D adalah dengan meletakkan semuanya pada file konfigurasi, biasanya adalah file `config.h`. Untuk dapat menggunakan metoda alternatif ini, mula-mula sisipkan instruksi `AC_CONFIG_HEADER(config.h)` pada `configure.in`. Selanjutnya, jalankan autoheader sebagai berikut:

```
autoheader
```

Perintah di atas akan menghasilkan `config.h.in` yang merupakan template untuk menghasilkan `config.h`

Sekarang modifikasi source code program untuk meng-include-kan file `config.h`. Menyambung contoh sebelumnya, maka `test.c` sekarang menjadi:

```
#ifdef HAVE_CONFIG
#include <config.h>
#endif
#include <stdio.h>
int main(int argc, char*argv[])
{
  printf ("Hello, World !\n");
#ifdef WORDS_BIGENDIAN
  printf("Big Endian\n");
#endif
```

```
}
```

Setelah tahap ini selesai, jalankan kembali autoconf dan automake. Proses `configure`, selain menghasilkan `Makefile` dari `Makefile.in`, juga memproses `config.h.in` untuk menghasilkan `config.h`, yang akan berisi:

```
/* config.h.  Generated automatically by configure.  */
/* config.h.in.  Generated automatically from configure.in by autoheader.
*/

/* Define if your processor stores words with the most significant
   byte first (like Motorola and SPARC, unlike Intel and VAX).  */
#define WORDS_BIGENDIAN 1
```

Hasil akhirnya adalah sama. Namun tentu saja meletakkan semua definisi di file `config.h` akan sangat menolong karena memudahkan debugging

Tanya Jawab (alias FAQ)

Mengapa harus menggunakan autoconf dan automake ?
Tidak harus, kok :-> Kedua utility ini hanya berfungsi sebagai alat bantu untuk memudahkan dan tidak harus dipakai.

Utility `aclocal` tidak bisa dijalankan. Mengapa ?
Pastikan bahwa Perl terinstal pada sistem Anda.

Waktu menjalankan automake, tampil pesan kesalahan karena file `install-sh` tidak ada ? Bagaimana solusinya ?
File `install-sh` adalah sebuah script biasa. File ini dibutuhkan untuk proses konfigurasi. Bila Anda tidak mempunyai file ini, silakan copy dari program lain. Lazimnya, program yang dipaket menggunakan automake dan autoconf selalu menyertakan file ini.

Pada saat automake, muncul beberapa pesan "required file ... not found", tetapi automake tidak mau membuat file tersebut secara otomatis. Mengapa demikian ?
Gunakan option `-a` saat menjalankan automake, jadi jangan jalankan '`automake`' tetapi '`automake -a`'.

Bagaimana mengatasi masalah yang muncul apabila program saya menggunakan bison.
Kemungkinan Anda lupa salah satu (atau kedua-duanya): menyisipkan relasi `AC_PROG_YACC` di `configure.in` dan menambahkan source untuk bison (biasanya file dengan ekstensi .y) ke bagian `SOURCES` di `Makefile.am`

Bisakah autoconf digunakan secara independen terhadap automake?
Bisa saja. Tentu `Makefile` yang dibuat sendiri harus mengenal substitusi variabel yang akan dilakukan oleh `configure`. Silakan merujuk ke manual penggunaan autoconf untuk lebih jelasnya.

Referensi

Trolltech http://www.trolltech.com
K Desktop Environment http://www.kde.org
Dokumentasi Qt http://doc.trolltech.com
GIMP http://www.gimp.org
GNU Make http://www.gnu.org/software/make/make.html
Situs resmi GNOME - www.gnome.org
KDE untuk developer - http://developer.kde.org
KDevelop - www.kdevelop.org
Qt dari Trolltech - http://www.trolltech.com/products/qt/
GNU autoconf Manual - http://www.gnu.org/manual/autoconf-2.13/autoconf.html
GNU automake Manual - http://www.gnu.org/manual/automake-1.3/automake.html
GNU Macro Processor Manual - http://www.gnu.org/manual/m4-1.4/m4.html
C-Scene: Multi-File Projects and the GNU Make Utility - http://www.syclus.com/cscene/CS2/CS2-10.html
Learning autoconf and automake - http://www.amath.washington.edu/~lf/tutorials/autoconf/
Learning the GNU Development tools - http://www.st-andrews.ac.uk/~iam/docs/tutorial.html

Biografi dan Profil

Ariya Hidayat. Menyelesaikan studi program S1 Teknik Fisika, ITB pada tahun 1999 dan program S2 Instrumentasi dan Kontrol, ITB pada tahun 2003. Saat ini tengah menggeluti riset doktor di bidang komunikasi optik di Universitas Paderborn, Jerman. Sempat aktif menyumbang tulisan untuk majalah InfoLINUX. Sejak tahun 2001 terlibat langsung sebagai developer KDE, bertanggung jawab terutama untuk pengembangan aplikasi KOffice.

Informasi lebih lanjut tentang penulis ini bisa didapat melalui:
Email: ariya@kde.org
URL: http://ariya.pandu.org