Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

redakce nakladatelství BEN – technická literatura
redakce@ben.cz

První program v Kylixu

Obr. 1 Program u01.dpr

Co to dělá:

Skoro nic, ale se vší parádou.

Jak:

Menu **File**/**New Application**. Menu **Run**/**Run**.

Vysvětlení:

Jednou z vlastností objektového programování je, že se nemusíme starat o věci, které nás nezajímají. Tedy že nezačínáme programovat z prázdna, ale upravujeme to, co již existuje. Většina věcí má nějaké implicitní hodnoty nebo chování a my se o ně začneme starat pouze tehdy, když s tímto implicitním řešením nebudeme spokojeni.

Tato aplikace tedy představuje „implicitní program“, umí zobrazit své okno, umí nechat změnit svoji velikost i polohu a umí se i nechat zavřít – čímž ukončí běh programu, uvolní paměť atd. atd. – prostě udělá všechny ty věci, o které se většinou nechceme zajímat a vyhovuje nám, že je to nějakým standardním způsobem vyřešeno.

Poznámky:

*Okno má implicitní chování i implicitní vlastnosti – velikost, název (**Form1**), nadpis, implicitní je i název programu (**Project1**).*

*Programy v Kylixu potřebují ke svému běhu celou řadu knihoven. Některé z nich se dají začlenit do výsledného programu (z menu **Project/Options** vyvoláme dialog, v němž zrušíme zaškrtnutí políčka **Packages/Build with runtime Packages**), některé nikoli.*

*Pokud máte nainstalovaný Kylix a naše programy budete spouštět přímo z jeho integrovaného vývojového prostředí (IDE), nebudete se o knihovny muset starat. Pokud budete pouštět programy sami z příkazové řádky shellu, budete muset nastavit cestu ke knihovnám do systémové proměnné **LD_LIBRARY_PATH**:*

```
LD_LIBRARY_PATH=/usr/local/kylix2/bin
export LD_LIBRARY_PATH
```

*(za předpokladu standardní instalace Kylixu 2 do adresáře **/usr/local/kylix2**).*

*Pokud Kylix nemáte nainstalovaný a pouštíte si naše přeložené programy, doporučujeme nejprve spustit instalační shell-skript **install.sh** (najdete ho na CD v adresáři **KvP**), který zkopíruje resp. nalinkuje potřebné soubory do adresáře **/tmp**. Proměnná **LD_LIBRARY_PATH** potom bude muset mít hodnotu **/tmp/KvP/lib**.*

*Pokud chcete správně vidět a používat českou diakritiku, budete si muset rovněž nastavit proměnnou **LANG**:*

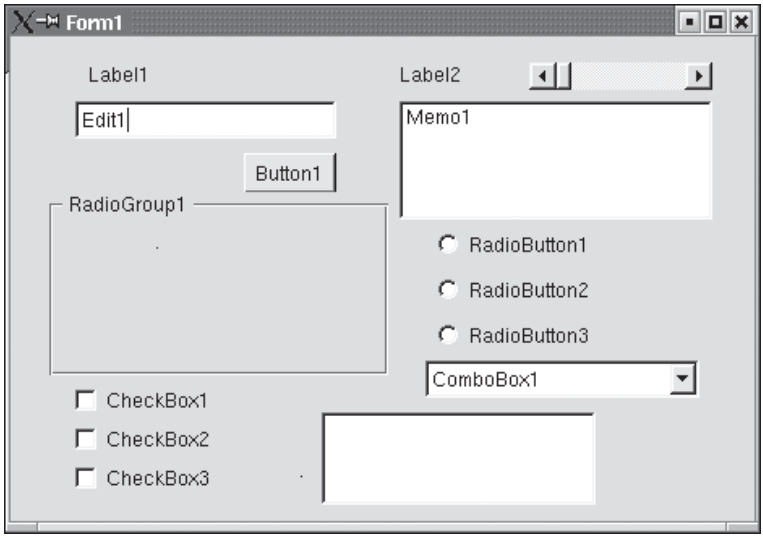
```
LANG=czech
export LANG
```

Ani to však nemusí stačit. Kylix je totiž velmi citlivý na přesné nastavení češtiny v X Window a i ve zdánlivě stejných prostředích se nám občas programy chovaly jinak (např. nezobrazovaly menu s českými písmenky).

*Kylix vychází z jiného vývojového prostředí firmy Borland určeného pro platformu MS Windows, z **Delphi**. Tato dvě prostředí jsou si velmi podobná, nicméně jsou mezi nimi mírné odlišnosti. Pokud na ně narazíme v našich příkladech, zmíníme se o tom v poznámce. (Abychom byli zcela korektní, Delphi od verze 6 mají dvojí knihovny – VCL a CLX – přičemž CLX je prakticky shodná s knihovnami Kylixu. Budeme-li tedy nadále hovořit o Delphi, budeme mít na mysli VCL knihovny.)*

A když už je řeč o verzích... Začali jsme naše pokusy s verzí 1, ale v té bylo ještě dost chyb. Proto jsme přešli na verzi 2, jakmile byla uvolněna. První dva díly knihy jsme ladili s Open Kylixem (volně šiřitelnou verzí), pro zbylé dva díly tato verze nepostačuje. Příklady jsme proto připravovali v tzv. Trial Kylixu (časově omezené plné verzi), definitivní překlad pro účely knihy jsme provedli neomezenou plnou verzí, kterou nám zapůjčila firma Borland (děkujeme!). V době vydání knihy se objevila verze 3, zkusili jsme v ní přeložit všechny příklady (případné změny oproti verzi 2 jsou v poznámkách), nicméně příklady jsme ponechali ve verzi 2. Trial verze Kylixu 3 je součástí CD přiloženého ke knize.

Komponenty



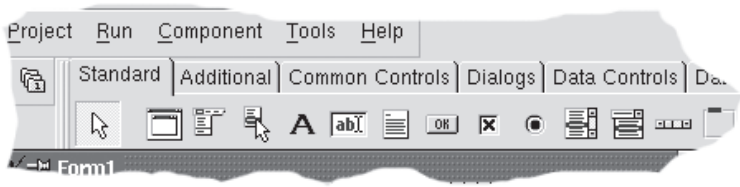
Obr. 2 Program u02.dpr

Co to dělá:

Pořád nic užitečného, ale vypadá to už opravdu opravdově.

Jak:

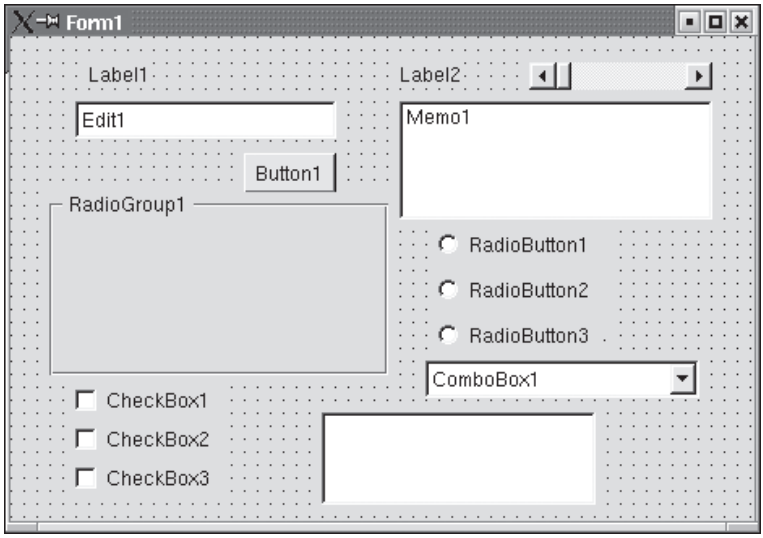
V horní části obrazovky, v hlavním okně Kylixu (pokud jste si ho nepřemístili), leží **Paleta komponent** (pokud jste ji neschovali – skoro všechno se dá změnit, takže budeme mluvit o standardním nastavení a standardních hodnotách a ta slova „pokud…“ si doplňujte sami).



Obr. 3 Paleta komponent

Jednotlivé ikony na jednotlivých stránkách Palety komponent představují stavební prvky programů v Kylixu. Jsou to objekty sloužící jako ovládací prvky nebo provádějící nějakou specializovanou činnost, mohou být viditelné i neviditelné a jejich umístěním do okna **Návrháře formuláře** (Form designer, viz *obr. 4*) říkáme, že mají být obsaženy v okně výsledného programu.

Umístěné komponenty potom, jakožto objekty, už mají své **vlastnosti** (property – zajímavá věc, později – prozatím něco jako proměnná), **metody** (souhrnné označení pro



Obr. 4 Návrhář formuláře

procedury a funkce) a umí reagovat na některé **události** (events). Například když jsme v Návrháři formuláře do svého okna umístili komponentu **Edit1** typu **TEdit**, tato komponenta umí reagovat na události související s editací textu – dovolí uživateli psát, vybírat úseky textu i používat schránku (clipboard). Podobně je tomu s ostatními komponentami.

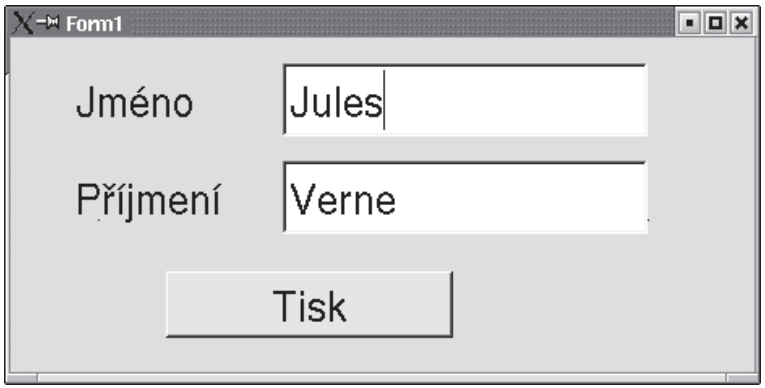
Vysvětlení:

Programování pod X Window je **programování řízené událostmi**.

Typický program se totiž v tomto prostředí nechová tak, že nejdříve načte veškerá vstupní data (a nepřestane, dokud je nedostane), pak provede výpočet a pak data vytiskne. Uživatel může aplikaci ovládat různými asynchronními zásahy (pohne myší, stiskne klávesu) a program musí umět reagovat na tyto **události**. Události přitom mohou být jednoduché, odpovídající elementárním vstupním jevům (třeba právě stisk klávesy nebo pohyb myši) i složité (vnitřní příkazy, zprávy) a programy potom pracují tak, že jen přijímají a zpracovávají události. Např. několik událostí pohybu a klikání myší v místech, kde je menu programu, vyústí ve složitější událost odpovídající položce z menu (třeba načíst data ze souboru), na tuto událost reaguje program tím, že skutečně přečte data ze souboru a uloží si je do svých proměnných, další klikání nebo stisky kláves mohou vyvolat událost „provést výpočet“ (součástí zpracování této události by měla být i kontrola, zda již byla načtena data)…

Takže programy obvykle v nějaké smyčce čekají na událost, potom ji zpracují a čekají dál – a mezi tím mohou totéž dělat ostatní programy pracující na stejném počítači. Psaní takových programů tedy obvykle sestává z **popisu reakcí na události**.

Vlastnosti komponent



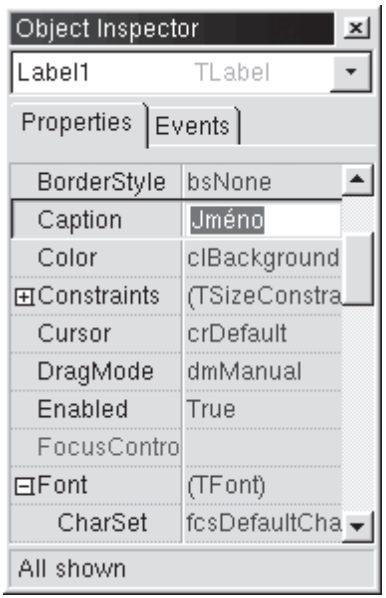
Obr. 5 Program u03.dpr

Co to dělá:

Stále nic, ale komponenty mají nastaveny některé vlastnosti.

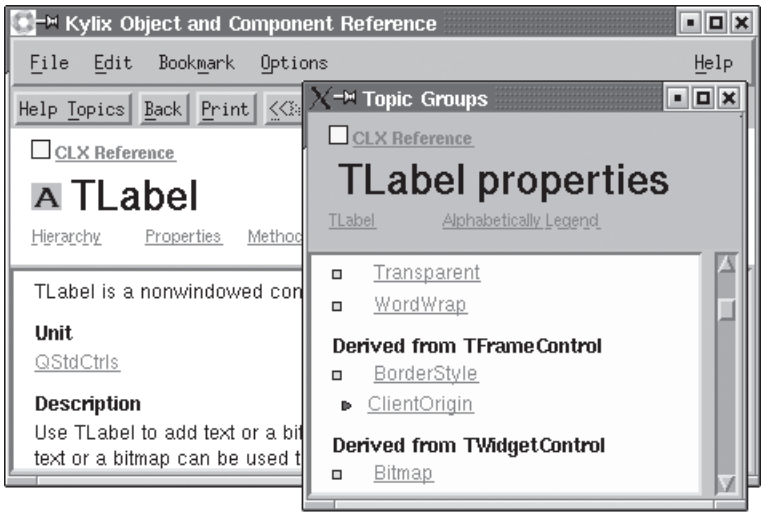
Jak:

Inspektorem objektů lze prohlížet i měnit vlastnosti vybrané komponenty nebo okna.



Obr. 6 Inspektor objektů – vlastnosti

Seznam vlastností, metod nebo událostí lze nalézt v nápovědě (vybereme komponentu a stiskneme **F1**).



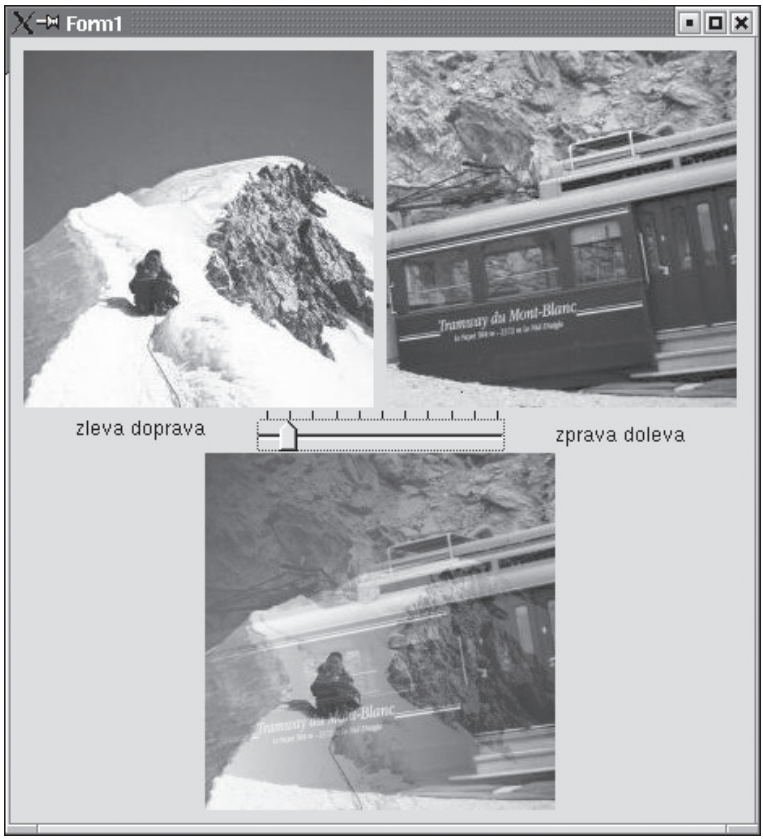
Obr. 7 Nápověda v Kylixu

Poznámky:

*Pokud se vám v posledních příkladech komponenty „nevešly“ do okna, máte nastaveno jiné rozlišení obrazovky, než bylo na počítači, kde se program vytvářel. Formulář hlavního okna si totiž pamatuje původní rozlišení (vlastnost **PixelsPerInch**) a přitom má nastavenu vlastnost **Scaled**, která říká, že se při běhu má program pokusit přizpůsobit aktuálnímu rozlišení. U oken s grafickými prvky to ale většinou není žádoucí, jelikož se tím mění proporce celého okna kromě obrázků. Napříště proto budeme tuto vlastnost nastavovat na **FALSE**. Současně je ovšem nutno nastavit na **FALSE** rovněž vlastnost **ParentFont**, aby do (nezvětšeného) okna nebyly umístěny (zvětšené) fonty X Window. Pokud byste si přáli, aby se okno jinému rozlišení přizpůsobovalo, ponechali byste obě vlastnosti nastaveny na **TRUE**, ale museli byste podle aktuálního rozlišení přizpůsobovat velikost okna.*

Rovněž některé další vlastnosti okna (minimální a maximální rozměr, tvar a ovládací prvky okraje, umístění při startu aj.) lze nastavit pomocí vlastností hlavního formuláře. Ovšem např. přesný tvar okraje a jeho ovládacích prvků závisí na používaném X Window Manageru. Ten rovněž může některé funkce zakázat (např. maximalizaci okna při startu).

Průměrný obrázek



Obr. 72 Program g33.dpr

Co to dělá:

Barvy bodů výsledného obrázku jsou (váženým) průměrem odpovídajících bodů dvou výchozích obrázků.

Jak:

Do komponent **Image1** a **Image2** vložíme obrázky, bitovou mapu pro komponentu **Image3** připravíme ve **FormCreate** (tak jako jsme to doteď dělali pro **Image2**).

Do formuláře umístíme komponentu **TrackBar1** ze záložky **Common Controls** a přiřadíme jí obslužnou metodu pro událost **Change**.

```
procedure TForm1.TrackBar1Change(Sender: TObject);
var
  x, y: Integer;
  p, q, t: Real;
  P1, P2, P3: PByteArray;
```

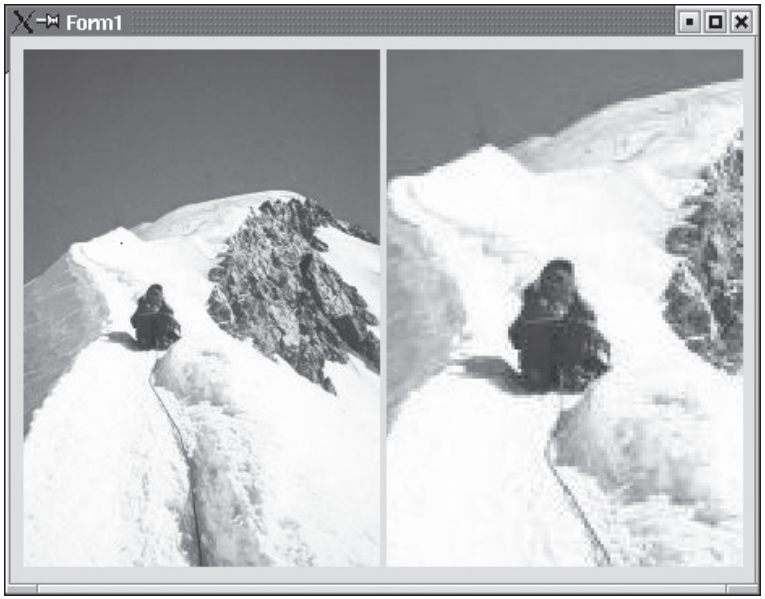
```
begin
  for y := 0 to Image1.Height-1 do
  begin
    P1 := Image1.Picture.Bitmap.ScanLine[y];
    P2 := Image2.Picture.Bitmap.ScanLine[y];
    P3 := Image3.Picture.Bitmap.ScanLine[y];
    for x := 0 to Image1.Width-1 do
    begin
      q := x/(Image1.Width-1);
      p := TrackBar1.Position/TrackBar1.Max;
      t := p*q+(1-p)*(1-q);

      setRGB( P3, x,
        Round( t*getRGB( P1, x ).r
          +(1-t)*getRGB( P2, x ).r ),
        Round( t*getRGB( P1, x ).g
          +(1-t)*getRGB( P2, x ).g ),
        Round( t*getRGB( P1, x ).b
          +(1-t)*getRGB( P2, x ).b ) );
    end;
  end;

  Image3.Repaint;
end;
```

Vysvětlení:

Ovládací prvek je komponenta typu **TTrackBar**, její chování i obsluha jsou analogické komponentě typu **TScrollBar**. Nastavením ovládacího prvku můžeme volit různé možnosti prolínání (od „zleva doprava“ přes „všude stejně“ až po „zprava doleva“).

Proměnná **t** určuje poměr zastoupení barev původních obrázků ve výsledku. Počítá se z relativní polohy jezdce ovládacího prvku (proměnná **p**) a z relativní pozice právě zobrazovaného sloupečku obrázku (proměnná **q**).

Zvětšenina



Obr. 73 Program g34.dpr

Co to dělá:

Po stisku tlačítka myši vytvoří zvětšený výřez se středem v místě stisku.

Jak:

Nemění barvu, ale mění souřadnice. Jako vzor pro bod [**x2**, **y2**] se bere bod [**x1**, **y1**].

```
procedure TForm1.Image1MouseDown(Sender: TObject;
  Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  x1, y1, x2, y2: Integer;
  P1, P2: PByteArray;
begin
  { stred vyrezu nesmi byt moc blizko u okraje }
  if X < Image1.Width div 4 then
    X := Image1.Width div 4;
  if X > 3*(Image1.Width div 4) then
    X := 3*(Image1.Width div 4);

  if Y < Image1.Height div 4 then
    Y := Image1.Height div 4;
  if Y > 3*(Image1.Height div 4) then
    Y := 3*(Image1.Height div 4);
```

```
    for y2 := 0 to Image2.Height-1 do
    begin
      y1 := Y+((y2-(Image1.Height div 2)) div 2);
      P1 := Image1.Picture.Bitmap.ScanLine[y1];
      P2 := Image2.Picture.Bitmap.ScanLine[y2];

      for x2 := 0 to Image2.Width-1 do
      begin
        x1 := X+((x2-(Image1.Width div 2)) div 2);
        getRGB( P2, x2 )^ := getRGB( P1, x1 )^;
      end;
    end;

    Image2.Repaint;
  end;
```

Vysvětlení:

Napsání správného vzorce pro výpočet **x1** z **x2** (a též pro **y1** z **y2**) je poněkud obtížné, protože namísto vymýšlení, kam se který bod **zobrazí**, si musíme představit, který bod je **vzorem** pro právě zpracovávaný bod. V tomto příkladu musíme vzít vzdálenost od středu obrázku (***x2*** – *polovina šířky*), tu vydělit dvěma (protože děláme detail) a přičíst k souřadnici středu výřezu.