systemove programovanie
gtk programovanie

15/03/2011

vyvoj okienkovych aplikacii: win32

* Win32 api [ciste C]
  * prilis nizkourovnove
    * vid posledna prednaska zo systemka minuly semester
* MS Foundation Classes (MFC) [C++]
  * C++ wrapper okolo Win32 api
* Windows Forms [.NET]
  * .NET wrapper okolo Win32 api
  * vyuziva volania z GDI
* Windows Presentation Foundation
  * reboot: postavene nad DirectX
  * XML, SVG, ...

vyvoj okienkovych aplikacii: linux

- GTK+ [defaultne ciste C]
  - povodne pre ucely GIMPu
  - dnes default pre GNOME
- Qt [defaultne C++]
  - produkovane Nokiou
  - dnes default pre KDE
- oba su multiplatformne
- binding pre mnohe jazyky

gtk+ hello world

```
#include <gtk/gtk.h>

int main (int argc, char * argv[]) {
   GtkWidget * window;

   gtk_init (&argc, &argv);
   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
   gtk_window_set_title (GTK_WINDOW (window), "Ahoj!");

   gtk_widget_show_all (window);

   gtk_main ();

   return 0;
}
```

gtk+ hello world

```
#include <gtk/gtk.h>

int main (int argc, char * argv[]) {
   GtkWidget * window;

   gtk_init (&argc, &argv);
   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
   gtk_window_set_title (GTK_WINDOW (window), "Ahoj!");

   gtk_widget_show_all (window);

   gtk_main ();

   return 0;
}
```

okno

inicializacia GTK+

gtk+ hello world

```
#include <gtk/gtk.h>

int main (int argc, char * argv[]) {
    GtkWidget * window;

    gtk_init (&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Ahoj!");

    gtk_widget_show_all (window);

    gtk_main ();

    return 0;
}
```

vytvorenie ,,instancie" okna a nastavenie titulku

gtk+ hello world

hlavickove subory s GTK

```
#include <gtk/gtk.h>

int main (int argc, char * argv[]) {
    GtkWidget * window;

    gtk_init (&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Ahoj!");

    gtk_widget_show_all (window);

    gtk_main ();

    return 0;
}
```

zobrazenie okna a jeho podkomponentov

spustenie event loopu

Ako kompilovat?

- uistime sa, ze mame libgtk
- na debiane:
  - aptitude install libgtk2.0-dev

```
gcc -Wall -g hellogtk.c -o hellogtk \
`pkg-config --cflags gtk+-2.0`\
`pkg-config --libs gtk+-2.0`
```

  - pkg-config prida do prikazu pre gcc prislusne odkazy na headre a kniznice GTK

PseudoOOP v GTK

- GtkWidget * window
  - widget sa tvari ako objekt
  - ale nema metody
- existuje sada funkcii s dobrou mennou konvenciou
  - gtk_window_set_title (GTK_WINDOW (window), "Ahoj!")
  - na okne (window) nastav (set) titulok (title)

PseudoOOP v GTK

- GtkWidget * window
- gtk_window_set_title ocakava parameter typu GtkWindow *
- okno JE widgetom: dedicnost
- ale v C dedicnost nie je
  - vsetko su pointery
- pretypovavanie sa riesi makrami
- GtkWidget ==> GtkWindow: makro GTK_WINDOW()

gtk+ hello world s tlacidlom

```
GtkWidget * button;
/* ... */

button = gtk_button_new_with_label("Klikni!");
gtk_container_add(GTK_CONTAINER(window), button);
```

signaly: ohlasuju udalosti

- gtk je udalostami riadeny framework
- spime v gtk_main() dovtedy, kym nenastane udalosti
- widget, na ktorom sa vyvolala udalost, vysle signal
  - pozor, toto nie su systemove signaly z UNIXu!
- nasledne sa zavola prislusna funkcia, ktora signal obsluzi

ako prepojit signaly s funkciou?

```
g_signal_connect(window, "destroy",
                 G_CALLBACK (gtk_main_quit), NULL);
```

- g_signal_connect() = prepajac
- window = signalizujuci komponent
- "destroy" = id signalu
- gtk_main_quit = obsluzna funkcia

- ak okno ma byt zavrete, zavola sa defaultna GTK obsluzna funkcia, ktora ukonci main loop

prepojenie tlacidla s funkciou

signalizujuce tlacidlo

```
void button_on_click(GtkWidget * widget,
                     gpointer data)
{
  g_print("Klik!");
}
```

pouzivatelske data

```
button = gtk_button_new_with_label("Klik!");
g_signal_connect(G_OBJECT(button),
                 "clicked",
                 G_CALLBACK(button_on_click),
                 NULL);
```

typ signalu

signalizujuce tlacidlo

obsluzna funkcia

pouzivatelske data do fcie

prepojenie tlacidla s funkciou

```
g_signal_connect(G_OBJECT(button),
                 "clicked",
                 G_CALLBACK(button_on_click),
                 NULL);
```

```
void button_on_click(GtkWidget * widget,
                     gpointer data)
{ ... }
```

poziciovanie komponentov

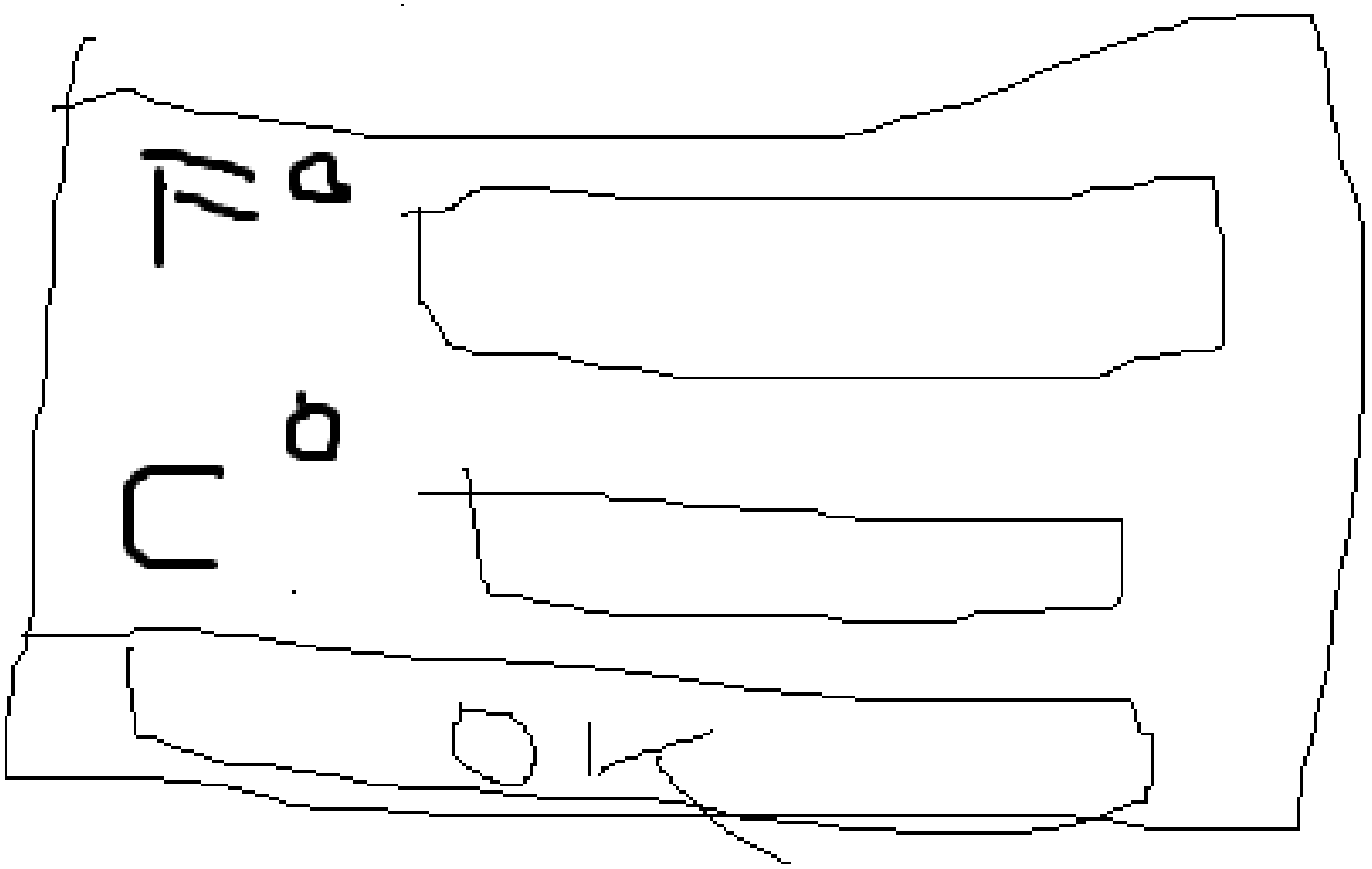
- co s viacerymi komponentami v okne?
- napr. vo win32: absolutne poziciovanie
  - kazdy komponent ma presne pixelove koordinaty polohy
  - zlo!
    - biedna internacionalizacia (i18n)
    - biedne zmeny velkosti
    - biedna odolnost voci zmenam dpi a pisma

gtk: boxy!

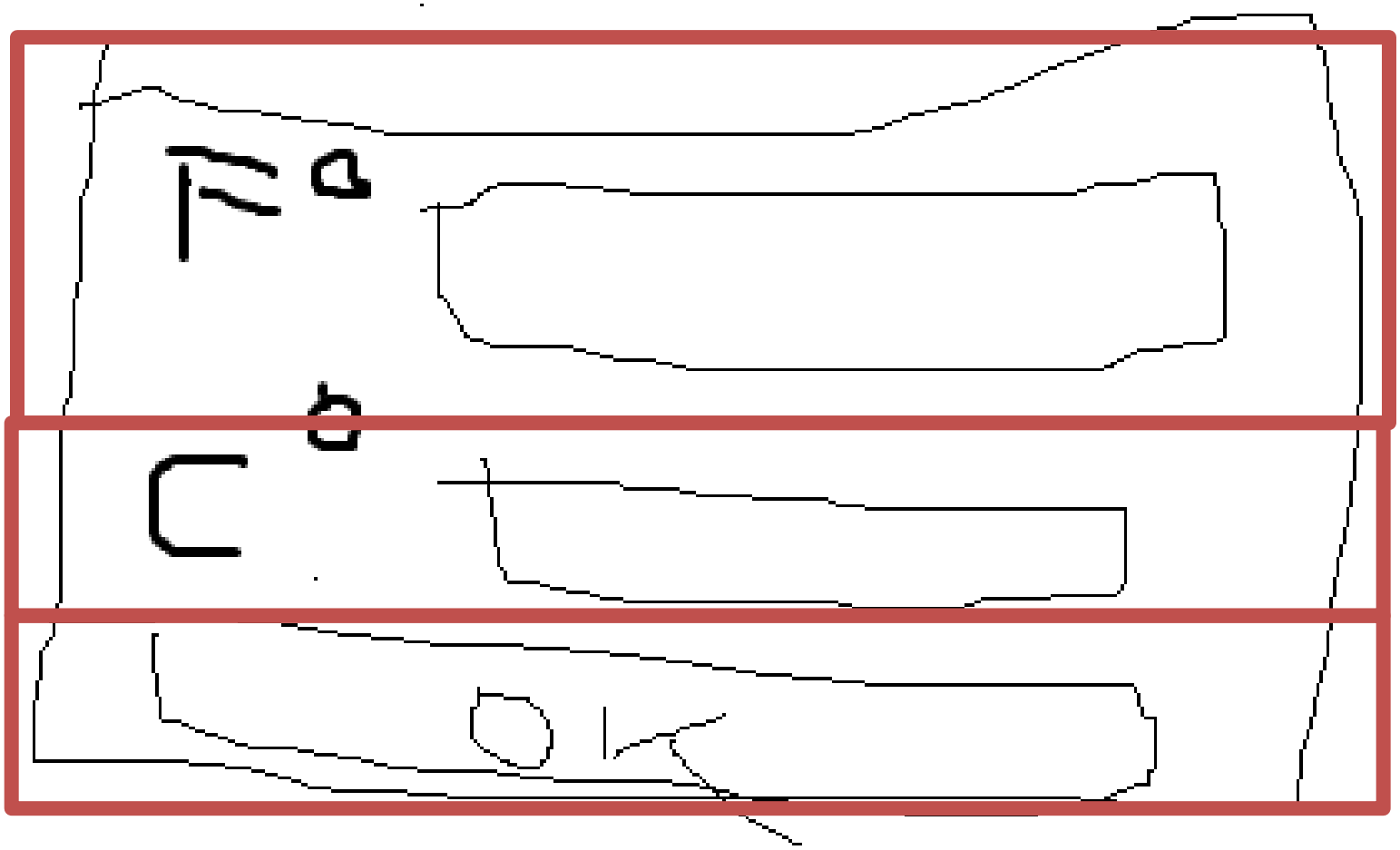
boxy a balenie

- hbox: horizontalny kontajner
  - komponenty sa ukladaju zlava doprava
  - ,,sadzac bez zalamovania"
  - ako
- vbox: vertikalny kontajner
  - komponenty sa ukladaju zhora nadol
  - ako
- hen k tabuli kreslit gui!

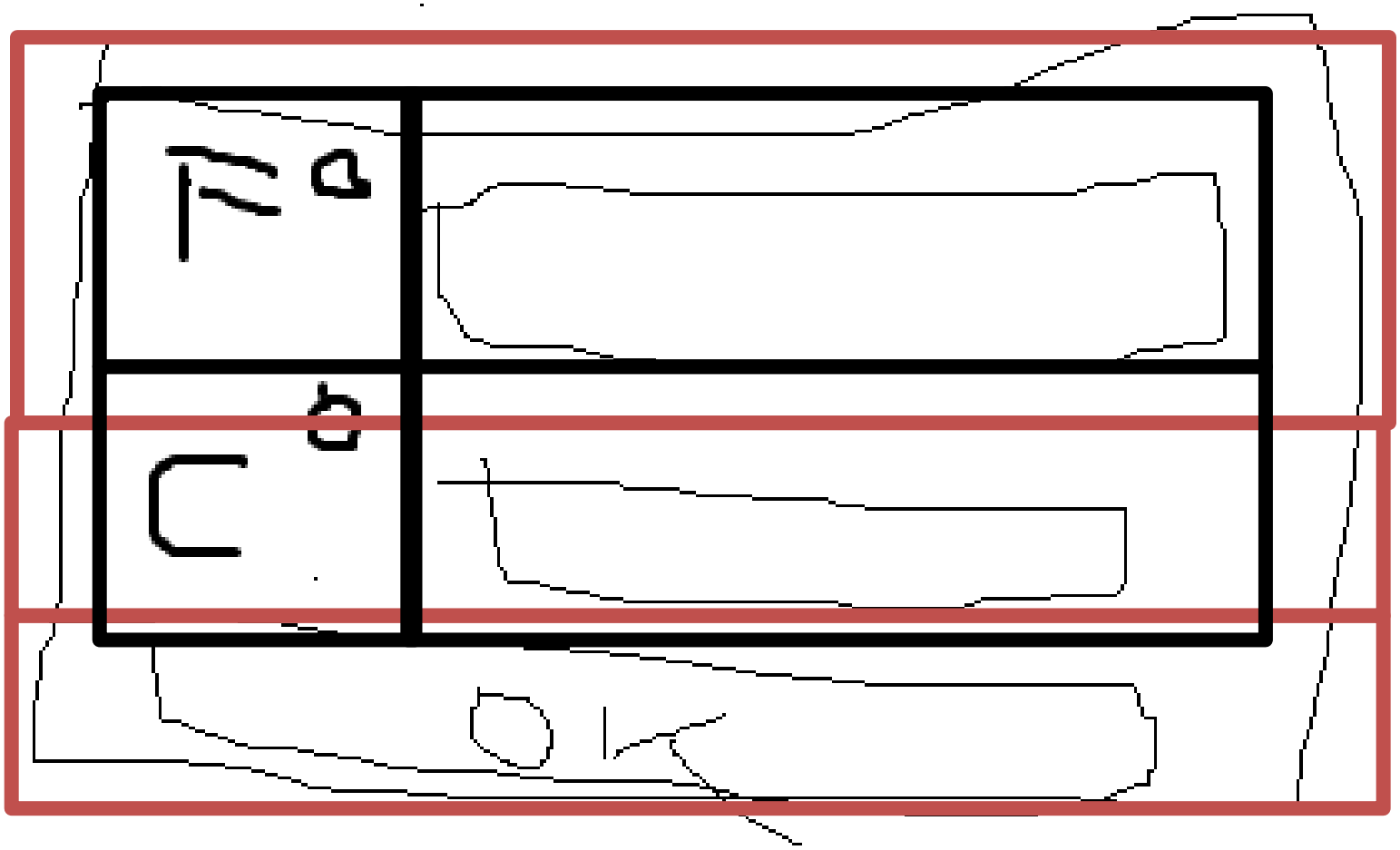
gui



gui



gui



gtk_box_pack_start

```
gtk_box_pack_start(GTK_BOX(fahrenheit_box),
                   label_fahrenheit,
                   FALSE, FALSE, 5);
```

- prida komponent do boxu
- 1: cielovy box
- 2: komponent
- 3: ma sa chlievik natahovat?
- 4: ma sa komponent v nom natahovat? ak nie, pridava sa vzduch
- 5: velkost vnutornej vypchavky

najprv len horny box

```
GtkWidget * window, * fahrenheit_box,
          * label_fahrenheit, * entry_fahrenheit;
/* ... */
```

```
fahrenheit_box = gtk_hbox_new(FALSE, 5);
label_fahrenheit = gtk_label_new("F:");
gtk_box_pack_start(GTK_BOX(fahrenheit_box),
                   label_fahrenheit,
                   FALSE, FALSE, 5);
```

```
entry_fahrenheit = gtk_entry_new_with_max_length(10);
gtk_box_pack_start(GTK_BOX(fahrenheit_box),
                   entry_fahrenheit,
                   FALSE, FALSE, 5);
```

```
gtk_container_add(GTK_CONTAINER(window),
                  fahrenheit_box);
```

dolezite je sa nezamotat

- kreslime si na papier
- komponenty oznacujme rozumnymi nazvami premennych
- doplnok:

```
gtk_box_pack_end(GTK_BOX(fahrenheit_box),
                 label_fahrenheit,
                 FALSE, FALSE, 5);
```

- pridava od konca boxu

natahovanie komponentov

```
gtk_box_pack_start(GTK_BOX(fahrenheit_box),
label_fahrenheit, FALSE, FALSE, 5);

gtk_box_pack_start(GTK_BOX(fahrenheit_box),
entry_fahrenheit, TRUE, TRUE, 5);

gtk_box_pack_start(GTK_BOX(fahrenheit_box),
button_calculate, FALSE, FALSE, 5);
```

- label: fixna sirka (a vyska; ta sa pri labeli nikdy nemeni)
- entry sa bude natahovat do sirky (ale ma fixnu vysku)
- button: fixna sirka, ale vyska pri buttone nie je fixnuta

tabulkovy layout

- ak je formular pravidelny, je to ovela lepsie na tvorbu

	<= 0	<= 1	<= 2
^ 0			
^ 1			
^ 2			

```
table = gtk_table_new(2, 3, FALSE);
```

- tabulka: 2R, 3S, stlpce rozlicnej sirky

tabulky

```
table = gtk_table_new(2, 3, FALSE);
label_fahrenheit = gtk_label_new("F:");
gtk_table_attach(GTK_TABLE(table), label_fahrenheit,
                 0, 1, 0, 1,
                 0, 0, 5, 5);
```

hranice tabulky, ku ktorym je ukotvena

- ukotvenia komponentu
  - zlava k 0,
  - sprava k 1,
  - zhora k 0,
  - zdola k 1

	<= 0	<= 1	<= 2
^ 0	X		
^ 1			
^ 2			

tabulky

```
table = gtk_table_new(2, 3, FALSE);
label_fahrenheit = gtk_label_new("F:");
gtk_table_attach(GTK_TABLE(table), label_fahrenheit,
                 0, 1, 0, 1,
                 0, 0, 5, 5);
```

spravanie pri zvacsovani buny

- GTK_FILL | GTK_SHRINK | GTK_EXPAND
  - FILL: komponent mensi nez bunka ju celu vyplni
  - SHRINK: komponent vacsi nez bunka sa zmensi
  - EXPAND: tabulka vyplni vsetko miesto v kontajneri
- parametre mozno OR-ovat
- posledne dva parametre: padding
  - vzdialenost medzi obsahom a okrajom bunky