Uporaba FLTK pri poučevanju programiranja v C++

Robert Meolic, Bogdan Dugonik

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko

E-pošta: meolic@uni-mb.si, bogdan.dugonik@uni-mb.si

Using FLTK in the course about C++ programming

This paper gives three relatively short C++ programs. They demonstrate the building of a graphical user interface by using free cross-platform library FLTK. All three programs are described and can be useful for demonstration of the basic concepts of object-oriented programming. Indeed, when students master them, their knowledge and programming skills are good enough to understand and create complex object-oriented programs – and this is our main goal of teaching programming in the first year.

1. Uvod

Naš izziv je naučiti osnove jezika C++ študente prvih letnikov, ki imajo le osnovno predznanje in izkušnje iz programiranja. Ni nujno, da je to C, v slovenskih gimnazijah se na primer učijo vedno bolj popularnega skriptnega jezika Python in nekateri tudi Javo.

Motiv za poučevanje C++ in nekaj modernih pristopov bomo v glavnem povzeli po prosojnicah avtorja C++, ki spremljajo njegovo knjigo [1].

Zakaj C++:

- C++ je zelo razširjen programski jezik,
- C++ je splošen jezik in omogoča neposredno izražanje idej na različnih področjih,
- za C++ obstaja standard ISO, ki se ga v praksi zelo upošteva,
- C++ je na voljo skoraj vsepovsod,
- znanje C++ je skoraj neposredno uporabno tudi pri programiranju v Javi in C#, ki sta prav tako zelo razširjena in pomembna jezika (zaradi drugačne zasnove nista neposredna tekmeca C++ in ga zato ne bosta izpodrinila!),
- C++ je živ jezik in z občasnimi posodobitvami standarda sledi razvoju računalništva (pogosto celo usmerja razvoj!).

Kako C++:

- C++ poučujmo kot samostojen jezik, ne pa kot nadgradnjo jezika C in se po nepotrebnem ne navezujmo na tehnike programiranja v jeziku C (B. Stroustrup pravi, da je pristop „najprej C“ zapravljanje študentovega časa ter vodi do slabega stila programiranja v C++),
- tudi če je cilj predmeta objektno usmerjeno programiranje, najprej poskrbimo, da študenti dobro osvojijo proceduralni del jezika C++,
- uporabljajmo knjižnice (ne učimo odkrivati tople vode) vendar pa ne silimo v akademski pristop „od zgoraj navzdol“ s prehitro uporabo visoko nivojskih konceptov, dokler študenti ne pridobijo potrebnega znanja in izkušenj,
- uporabljene principe in gradnike vedno razložimo, tako da študenti zares razumejo njihov smisel ter delovanje, ne pa, da jih pokažemo le kot del „čarobnega“ recepta,
- uporabimo čim bolj realistično programersko okolje in čim bolj realistične primere ter se ne omejimo na teoretično okolje, v katerem vse deluje brez problema.

V tem članku še prav posebej izpostavljamo skrb za motivacijo študentov. Znanje programiranja danes ni pričakovani del splošne izobrazbe in tudi zanimanje študentov tehnike za to snov ni samo po sebi umevno. Najbolj moteč je razkorak med programi na vajah in programi, ki jih študenti vsakodnevno uporabljajo na računalniku. Medtem ko pri slednjih s pomočjo zmogljivih grafičnih vmesnikov in miške večino časa nekaj izbiramo in klikamo, so programi na vajah iz programiranja ponavadi omejeni na vnos s tipkovnico in izpis besedila. Ta razkorak je tako velik, da študenti skorajda ne čutijo več povezave med tem, kar se morajo učiti, in tem, kar v praksi uporabljajo (kar jih zanima). Kako naj bo programiranje privlačno, če študent po dveh semestrih programiranja ne more prijateljem pokazati nič drugega kot program, ki v „črnem oknu“ (terminalu) nekaj izpisuje? Še pognati se ga ne da na uveljavljeni način - s klikom na ikono.

Dejstvo, da C++ nima standardne grafične knjižnice (kot npr. Java), ne sme biti razlog, da se grafiki popolnoma odpovemo. Poleg izrednega motivacijskega učinka lahko naštejemo še veliko drugih prednosti (spet povzeto po B. Stroustrupu [1]):

- grafika je koristna, od programa za analizo podatkov na primer pričakujemo,da riše grafe,
- grafika vključuje veliko zanimive, poučne in tudi zahtevne kode,
- grafika je vir odličnih načrtovalskih vzorcev,
- grafika ponuja številne izzive, ki so zelo primerni za reševanje s tehniko objektno usmerjenega programiranja,

- grafika sama po sebi ni trivialna in se je zato veliko študentov samoiniciativno ne bo lotilo (posledično ne bodo našli pravega stika med vajami in realnimi programi na računalniku).

Ena od velikih prednosti C++ je prenosljivost kode med različnimi sistemi, ki pa seveda velja le, če uporabljamo prenosljive knjižnice. Žal sta oba najbolj uporabljana operacijska sistema na računalnikih komercialna in spodbujata uporabo lastnih (zaprtih) knjižnic. Ker njihova uporaba ogrozi prenosljivost programov (in hkrati študente sili k uporabi točno določenih komercialnih rešitev) se jim pri poučevanju izogibamo, s čimer pa se moramo odpovedati tudi številnim dobrim in v industriji razširjenim načrtovalskim orodjem. Na srečo je razvoj računalništva v zadnjem času prinesel tudi zmogljiva prenosljiva ogrodja za grafiko. Najbolj znana so FLTK [2], GTK+ [3], Qt [4] in wxWidgets [5]. Mi smo na vajah iz programiranja izbrali FLTK, ki je med naštetimi res najmanj zmogljivo ogrodje (in zato v realnosti najmanj prisotno), je pa zato preprosto in se lahko dobro vključi v vaje iz programiranja v C++.

2. FLTK

Fast Light Toolkit (FLTK) je prosta in prenosljiva knjižnica za izdelavo grafičnih vmesnikov s C++. Podpira vse grafične gradnike 2D in 3D (OpenGL). Poleg prenosljivosti (deluje na GNU/Linux, MS Windows, Mac OS X) ima še dobro lastnost, da je sorazmerno majhna in modularna ter tako tudi ob statičnem povezovanju ne povzroči ogromnih izvršilnih datotek. Junija 2011 je bila izdana stabilna različica v1.3.0. Zadnje posodobitve so omogočile uporabo UTF-8, s tem pa tudi šumnike in druge posebne znake.

Knjižnica FLTK je prilagojena za uporabo v C++. Potrebno je solidno predznanje C++, še posebej lahko izpostavimo, da je potrebno dobro razumeti:

- vključevanje knjižnic,
- konstruktorje in destruktorje,
- dedovanje razredov,
- inicializacijske sezname,
- reference in prenos po referenci.

Posebnosti, s katerimi se študenti verjetno prvič srečajo, pa so:

- gradniki GUI,
- razporejanje gradnikov GUI,
- dogodkovno programiranje.

Pomoč in zglede za FLTK najdemo na spletu. Izpostavimo lahko FLTK Programming Manual [6], FLTK Cheat Page [7] ter foruma na naslovih http://fltk.org/newsgroups.php in http://www.gidforums.com/f-48.html. Zanimiva demonstracija je FLTK Recycling game [8].

3. Vaja1

Program vaja1.cpp prikazuje dva gradnika grafičnih vmesnikov, in sicer okno (Fl_Window) ter polje za izpis besedila (Fl_Text_Display).

```
// vaja1.cpp - copyright Robert Meolic 2011
// Build with: g++ -Wall -o vaja1.exe vaja1.cpp -lfltk

#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Text_Display.H>
#include <string>  // delo z nizi
using namespace std;

class MainWindow : public Fl_Window {
  private:
  Fl_Text_Buffer outputBuffer; //besedilo v tekstovnem polju
  Fl_Text_Display mainDisplay; //tekstovno polje

  public:
  MainWindow(int x, int y); // navadni konstruktor
  void prikazi(string vrstica);
};

MainWindow::MainWindow(int x, int y):
    Fl_Window(x,y,"Glavno okno"), //velikost in ime okna
     mainDisplay(0,0,w(),h()) //velikost in polozaj tekstovnega po-
lja
{
  mainDisplay.buffer(outputBuffer); //povezi polje z besedilom
  resizable(mainDisplay); //omogoci prilagajanje velikosti
}

void MainWindow::prikazi(string vrstica) {
  outputBuffer.append(vrstica.c_str()); //dodaj besedilo
}

int main() {
  MainWindow mainWindow(800,400);
  mainWindow.prikazi("Pozdravljen svet!\n");
  mainWindow.show();
  return Fl::run();
}
```

Knjižnice za FLTK se privzeto namestijo v sistemsko mapo, zato pri vključevanju uporabimo špičaste oklepaje. Ker ima FLTK modularno zgradbo, moramo vključiti več zaglavij - vsa tista, ki jih uporabljamo. Razreda Fl_Text_Display in Fl_Text_Buffer sta med seboj povezana (Fl_Text_Display uporablja Fl_Text_Buffer), zato Fl_Text_Buffer.H ni treba posebej vključevati.

Razred MainWindow je izpeljan iz Fl_Window z metodo javne izpeljave. Potrebujemo vsaj en konstruktor,destruktor v tem preprostem primeru ni potreben. Objekt iz razreda Fl_Text_Display (v našem programu je to mainDisplay) je gradnik, ki prikazuje vsebino objekta iz razreda Fl_Text_Buffer (v našem programu je to outputBuffer). Ker je outputBuffer deklariran kot zaseben, smo dodali javno metodo prikazi(string vrstica), s katero lahko uporabnik spreminja prikazano besedilo (besedilo lahko le dodaja).

Oglejmo si najprej konstruktor za MainWindow. Ker razredi Fl_Window, Fl_Text_Display in tudi večina drugih iz knjižnice FLTK nimajo definiranih privzetih konstruktorjev, se ne moremo izogniti inicializacijskim seznamom. V konstruktorjih brez inicializacijskega seznama se namreč pri C++ za vse elemente najprej izvede privzeti konstruktor in šele potem se začne izvajati koda v telesu. Uporabljeni konstruktor za Fl_Window ima 3 parametre, s katerimi določimo velikost in ime okna. Uporabljeni konstruktor za Fl_Text_Display ima 4 parametre, s katerimi določimo velikost in položaj tekstovnega polja znotraj okna. Podamo absolutne koordinate, pri čemer je izhodišče (0,0) v zgornjem levem vogalu.

Definicija metode prikazi(string vrstica) ne potrebuje komentarja, uporabljena je pretvorba med nizom C++ in C, ki jo študenti verjetno že poznajo. Bolj zanimiv je glavni program, ki tik pred koncem kliče metodo Fl::run(). Ko uporabljamo GUI dobimo dogodkovno voden program. Za razliko od običajnega (premočrtnega)

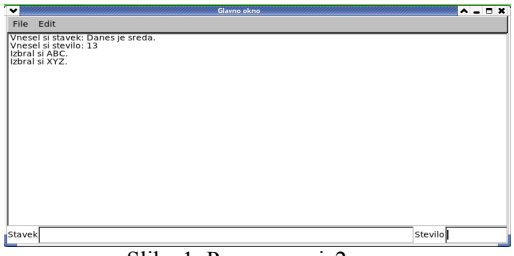
programa tukaj ni tako, da funkcija main() zaporedoma (seveda z uporabo vejitev in zank) izvede določene akcije in se potem konča, s tem pa je tudi konec programa. Pri programu z GUI glavni program poskrbi, da se na ekranu pokaže okno z neko vsebino, nato pa se čaka na dogodke (pri FLTK to izvedemo s klicem Fl::run()), kot je na primer klik na gumb. Vsak dogodek se (čimbolj učinkovito) obdela, nato pa se spet vrnemo v čakanje. Prva vaja še ne vsebuje obdelave dogodkov (želeli smo, da je prva vaja preprosta), jih boste pa našli v nadaljevanju. Zahtevnejši pristop je, ko želimo omogočiti nove dogodke, še preden se trenutni dogodek v celoti obdela (uporaba niti!), a to niso več osnove programiranja.

4. Vaja 2

Program vaja2.cpp (slika 1) doda tekstovne oznake (Fl_Output), polja za vnos (Fl_Input) in meni (Fl_Menu_Bar). Slednja dva sta aktivna gradnika, zato imata t. i. funkcije callback. To so metode, ki se izvedejo ob posameznem dogodku. Morajo biti statične (glej razlago pri tretji vaji). Mi v prvi vrstici vsake funkcije callback pridobimo kazalec na mainWindow, nato pa ustrezno obdelamo dogodek. Funkcijo callback registriramo s klicem metode callback(Fl_Callback *cb), razen pri meniju, kjer jo enostavno navedemo kot parameter metode add(...). Več o meniju in prilagajanju gradnikov s konstantami (npr. FL_FLAT_BOX) najdete v dokumentaciji.

```
// vaja2.cpp - copyright Robert Meolic 2010/2011
// Build with: g++ -Wall -o vaja2.exe vaja2.cpp -lfltk

#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Text_Display.H>
#include <FL/Fl_Menu_Bar.H>
#include <FL/Fl_Output.H>
#include <FL/Fl_Input.H>
#include <iostream> // vhodni in izhodni tokovi
#include <string>  // delo s stringi
#include <sstream>  // pretvorba med stevili in stringi
#include <cstdlib>  // funkcija exit()
using namespace std;

class MainWindow : public Fl_Window {
  private:
  Fl_Text_Buffer outputBuffer;
  Fl_Text_Display mainDisplay;
  Fl_Menu_Bar mainMenu;
  Fl_Output labelStavek,labelStevilo;
  Fl_Input inputStavek,inputStevilo;
  static void cbMenuAbc(Fl_Widget* w, void* data);
  static void cbMenuXyz(Fl_Widget* w, void* data);
  static void cbMenuExit(Fl_Widget* w, void* data);
  static void cbInputStavek(Fl_Widget* w, void* data);
  static void cbInputStevilo(Fl_Widget* w, void* data);

  public:
  MainWindow(int x, int y); // navadni konstruktor
  void prikazi(string vrstica);
};

MainWindow::MainWindow(int x, int y):
    Fl_Window(x,y,"Glavno okno"),
    mainDisplay(0,30,w(),h()-60),
    mainMenu(0,0,w(),30),
    labelStavek(0,h()-30,50,30),
    labelStevilo(w()-150,h()-30,50,30),
    inputStavek(50,h()-30,w()-200,30),
    inputStevilo(w()-100,h()-30,100,30)
{
  mainDisplay.buffer(outputBuffer);
  resizable(mainDisplay);
  mainMenu.add("File/Exit",0,cbMenuExit);
  mainMenu.add("Edit/ABC",0,cbMenuAbc);
  mainMenu.add("Edit/XYZ",0,cbMenuXyz);
  labelStavek.value("Stavek:");
  labelStavek.box(FL_FLAT_BOX);
  labelStevilo.value("Stevilo:");
  labelStevilo.box(FL_FLAT_BOX);
  inputStavek.when(FL_WHEN_ENTER_KEY);
  inputStavek.callback(cbInputStavek);
  inputStevilo.when(FL_WHEN_ENTER_KEY);
  inputStevilo.callback(cbInputStevilo);
}

void MainWindow::prikazi(string vrstica) {
  outputBuffer.append(vrstica.c_str());
}

void MainWindow::cbMenuExit(Fl_Widget* w, void* data) {
  exit(0); // takoj koncaj program
}

void MainWindow::cbMenuAbc(Fl_Widget* w, void* data) {
  MainWindow* mainWindow = ((MainWindow*) w->parent());
  mainWindow->prikazi("Izbral si ABC.\n");
}

void MainWindow::cbMenuXyz(Fl_Widget* w, void* data) {
  MainWindow* mainWindow = ((MainWindow*) w->parent());
  mainWindow->prikazi("Izbral si XYZ.\n");
}

void MainWindow::cbInputStavek(Fl_Widget* w, void* data) {
  MainWindow* mainWindow = ((MainWindow*) w->parent());
  string vnos = mainWindow->inputStavek.value();
  string izpis = "Vnesel si stavek: " + vnos + "\n";
  mainWindow->prikazi(izpis);
  mainWindow->inputStavek.value("");
}

void MainWindow::cbInputStevilo(Fl_Widget* w, void* data) {
  MainWindow* mainWindow = ((MainWindow*) w->parent());
  int stevilo;
  stringstream vnos(mainWindow->inputStevilo.value());
  vnos >> stevilo;  // pretvori niz znakov v stevilo
  stringstream izpisStevilo;
  izpisStevilo << stevilo; // pretvori stevilo v niz znakov
  string izpis = "Vnesel si stevilo: " + izpisStevilo.str() + "\n";
  mainWindow->prikazi(izpis);
  mainWindow->inputStevilo.value("");
}

int main() {
  MainWindow mainWindow(800,400);
  mainWindow.show();
  return Fl::run();
}
```

V konstruktorju za MainWindow dobro vidimo problem razporejanja gradnikov po ekranu. FLTK (zaenkrat) ponuja le absolutno razporejanje gradnikov s podajanjem koordinat. Lahko pa vsakemu gradniku z metodo resizable(Fl_Widget *o) določimo, ali bo njegova velikost prilagodljiva (ko zmanjšamo ali povečamo okno). Učinek najlažje dojamemo, če malo eksperimentiramo, v konstruktor za MainWindow na primer dodajte stavek resizable(labelStevilo) in si oglejte učinek.



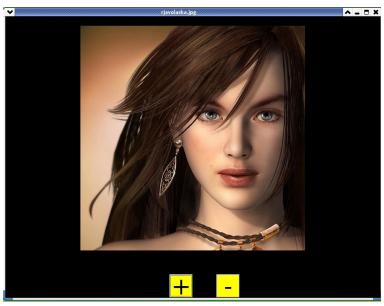
Slika 1: Program vaja2.cpp

5. Vaja 3

Program vaja3.cpp (slika 2) prikaže sliko iz datoteke (v našem primeru je to rjavolaska.jpg) in potem z dvema gumboma omogoči, da sliko povečujemo in zmanjšujemo. Novost je uporaba Fl_Image, Fl_Box in Fl_Button.

```
// vaja3.cpp - Copyright Robert Meolic 2010/2011
// Build with: g++ -Wall -o vaja3.exe vaja3.cpp -lfltk -lfltk_images

#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_JPEG_Image.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Button.H>
```

```
#include <iostream> // vhodni in izhodni tokovi
#include <string> // delo s stringi
using namespace std;

class MainWindow : public Fl_Window {
  private:
  Fl_Image* imageOrg;
  Fl_Image* imageShown;
  Fl_Box workingBox;
  Fl_Button buttonPlus,buttonMinus;
  int zoom;
  static void cbButtonPlus(Fl_Widget* w, void* data);
  static void cbButtonMinus(Fl_Widget* w, void* data);

  public:
  MainWindow(string imageName, Fl_Image& thisImage);
  ~MainWindow();
  void zoomIn();
  void zoomOut();
  void redrawImage();
};

MainWindow::MainWindow(string name, Fl_Image& image):
    Fl_Window(800,600,name.c_str()),
    imageOrg(&image),
    imageShown(NULL),
    workingBox(160,20,480,480),
    buttonPlus(350,550,50,50),
    buttonMinus(450,550,50,50),
    zoom(100)
{
  color(FL_BLACK); //ozadje okna
  workingBox.box(FL_FLAT_BOX);
  workingBox.color(FL_BLACK);
  resizable(workingBox);
  buttonPlus.label("+");
  buttonPlus.labelsize(50);
  buttonPlus.color(FL_YELLOW);
  buttonPlus.callback(cbButtonPlus);
  buttonMinus.label("-");
  buttonMinus.labelsize(50);
  buttonMinus.color(FL_YELLOW);
  buttonMinus.callback(cbButtonMinus);
}

MainWindow::~MainWindow() {
  delete imageShown;
}

void MainWindow::zoomIn() { //povecaj zoom
  if (zoom < 200) {zoom += 10; redrawImage();}
}

void MainWindow::zoomOut() { //zmanjsaj zoom
  if (zoom > 10) {zoom -= 10; redrawImage();}
}

void MainWindow::redrawImage() { //osvezi sliko
  delete imageShown;
  imageShown=imageOrg->copy(4.8*zoom,4.8*zoom);
  workingBox.image(*imageShown);
  workingBox.redraw();
}

void MainWindow::cbButtonPlus(Fl_Widget* w, void* data) {
    MainWindow* mainWindow = ((MainWindow*) w->parent());
    mainWindow->zoomIn();
}

void MainWindow::cbButtonMinus(Fl_Widget* w, void* data) {
    MainWindow* mainWindow = ((MainWindow*) w->parent());
    mainWindow->zoomOut();
}

int main() {
  Fl::visual(FL_DOUBLE|FL_INDEX);
  string imageName = "rjavolaska.jpg";
  Fl_JPEG_Image image(imageName.c_str());
  MainWindow mainWindow(imageName,image);
  mainWindow.redrawImage();
  mainWindow.show();
  return Fl::run();
}
```



Slika 2: Program vaja3.cpp

V glavnem programu tvorimo sliko kot objekt iz razreda Fl_JPEG_Image. MainWindow to sliko obravnava kot objekt iz razreda Fl_Image. To je možno, ker je Fl_JPEG_Image izpeljan iz Fl_Image. Vedno, ko neka metoda (tudi konstruktor) pričakuje Fl_Image mu lahko "podtaknemo" Fl_JPEG_Image. Seveda pa funkcija, ki pričakuje Fl_Image, sliko lahko obdela le na splošen način brez postopkov specifičnih za slike JPEG.

Vaja 3 skriva še nekaj poučnih podrobnosti. Za prvi test spremenite konstruktor za MainWindow tako, da zbrišete referenco (&) za Fl_Image (to je treba storiti na dveh mestih). Prevajalnik sporoči napako: *Fl_Image::Fl_ Image(const Fl_Image&)' is private*, a kaj ta napaka sploh pomeni? Odgovor: Fl_Image nima definiranega kopirnega konstruktorja!

Drugi test zahteva malo več sprememb. Ideja je, da deklaracijo slike prestavimo iz glavnega programa v konstruktor za MainWindow. Program se prevede brez napak, vendar se med delovanjem zruši. Kaj je narobe? Odgovor: ker je slika lokalna spremenljivka v konstruktorju se po njegovi izvedbi zbriše. Zato referenca shranjena v imageOrg kaže v prazno!

Pri tretjem testu zbrišemo static pred deklaracijo funkcij callback. Prevajalnik sporoči napako *no matching function for call to 'Fl_Button:: callback(<unresolved overloaded function type>)'* v vrstici buttonPlus. callback(cbButtonPlus). Zakaj to, če pa funkcija cbButtonPlus obstaja, ima pravilne argumente in je pravilno deklarirana kot void? Vzrok napake je, da prevajalnik loči med tipom kazalec na funkcijo, ki je navadni član razreda (v našem primeru void (MainWindow::*)(Fl_Widget*,void*)) in tipom kazalec na funkcijo, ki je statični član razreda (v našem primeru void (*)(Fl_Widget*,void*)). Slednji je istega tipa, kot če funkcija ne bi bila član nobenega razreda in takega zahteva funkcija callback.

Pa še izzivi za izboljšanje programa. V programu vse Fl_Window zamenjajte s Fl_Double_Window in se boste znebili nadležnega utripanja. Če namesto Fl_ Button uporabite Fl_Repeat_Button pa boste gumbe lahko "držali" pritisnjene.

6. Zaključek

Za razpravo ni prostora (niti potrebe). Če bo članek koristen, bo dosegel svoj namen. Vse tri programe dobite na internetu na http://lms.uni-mb.si/~meolic/pe2/.

Literatura

[1] B. Stroustrup. Programming: Principles and Practice Using C++. Addison-Wesley, 2009.
[2] Fast Light Toolkit. http://fltk.org/
[3] The GTK+ Project. http://www.gtk.org/
[4] QT - Cross-platform application and ... http://qt.nokia.com/
[5] wxWidgets - Cross-Platform GUI... http://wxwidgets.org/
[6] FLTK Programming Manual . http://www.fltk.org/doc-1.3/
[7] G. Ercolano. Erco's FLTK Cheat Page. http://seriss.com/people/erco/fltk/
[8] Y. D'Elia. FLTK Recycling Game! http://www.develer.com/~wavexx/regame/