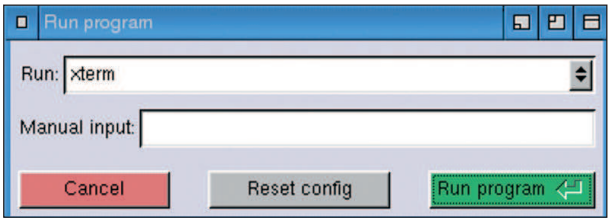
Programfejlesztés az eFluid segítségével

Sorozatunk utolsó fejezetében egy egyszerű programot fogunk elkészíteni az eFltk grafikus fejlesztőeszközének segítségével.

Gyakran előfordul, hogy bizonyos ablakkezelő alkalmazások használata közben egyszerű módon szeretnénk elindítani alkalmazásokat. Hasznos segédprogram lehet egy alkalmazások indítására alkalmas program, amely tárolja a korábban futtatott programok nevét, majd a következő indításnál már egy listából is kiválaszthatjuk a megfelelő elemet. A kész program képe az *1. ábrán* látható.

Mielőtt egy programot elkezdünk fejleszteni, mindenképpen érdemes néhány mondatban megfogalmazni, hogyan kell annak működnie. Jelen esetben célunk egy olyan program elkészítése, mely megkönnyíti egy egyszerű ablakkezelő futtatása mellett más programok indítását. A programot saját magamnak készítettem, az ***IceWM*** ablakkezelő ***„Futtatás...”*** menüpontjának megvalósítására. Ebben az ablakkezelőben a programok indítására alkalmazható bármilyen program, melyet a konfigurációs állományban megadunk az ablakkezelőnek. Elsődleges célom, hogy a program könnyen érthető legyen és emellett megvalósítsa a kitűzött célt.

A példában szereplő program kétféle fő funkciót valósít meg. Elsősorban az alsó mezőben megadott programot fogja elindítani, majd amennyiben az nem szerepel a felső listában, kilépés előtt tárolja az új listaelemmel kibővített programlistát. Ha nem írunk be semmilyen új programot a beviteli mezőbe, akkor a listából kiválasztott alkalmazást indíthatjuk el. A program konfigurációs állománya a felhasználó saját könyvtárában lévő ***.flrun.cfg*** állomány. Ebben az állományban minden sor egy-egy futtatható programot ír le. A példában szereplő program indulásakor beolvassa ezt az állományt, majd feltölti a listát a beolvasott sorokkal. A felhasználó ezután beírhat új elemet vagy választhat a már meglévők közül. A program felhasználói felületén mindössze három gomb segíti a használatot. Az alapértelmezett (ENTER billentyűvel működtethető) gomb indítja a kiválasztott vagy megadott alkalmazást. A második gomb az alkalmazásból való kilépésre szolgál és a főablak bal alsó sarkában kapott helyet. Amennyiben a konfigurációs állomány alapállapotba állítására van szükség, ezt a műveletet a középső gomb (***„Reset config”*** felirattal) segítségével végezhetjük el. Természetesen a ***Linuxban*** megszokott módon, tetszőleges szövegszerkesztő programmal is megváltoztathatjuk a beállításokat tároló állományt.



1. ábra Az flRun alkalmazás ablaka

A program készítése során fontos szempont volt számomra, hogy bemutassam az ***eFltk*** grafikus fejlesztőeszközének használatát, így nem használtam más szövegszerkesztőt vagy fejlesztői környezetet a megvalósítás során. Mindössze az ***eFluid***-ra és egy terminál ablakra volt szükségem, melyben a fordítást időnként elvégeztem, majd a programot elindítottam.

Lássunk tehát hozzá a megvalósítás lépéseinek. Miután kialakítottuk a felhasználói felület képét, hozzunk létre egy új függvényt. Ez az első lépés szükséges ahhoz, hogy a főprogram ablakát megtervezhessük és elhelyezhessük rajta a megfelelő elemeket. Tehát a fejlesztőkörnyezet menüjében keressük meg a ***„New”*** menüt és válasszuk ki a ***Code->Function/Method*** menüpontokat. A megjelenő párbeszédablakban hagyhatunk mindent az alapértelmezett beállításon. Ezután hozzunk létre egy új ablakot a ***New->Group->Window*** pontok választásával. Állítsuk be a méretét, majd kattintsunk rá kettős kattintással. Itt megváltoztathatjuk például az ablak háttérszínét vagy címét, de leginkább a ***„C++”*** fülecskén lévő adatok megadása fontosabb.

Az ablak neve ebben a példában `MainWindow` és az ablak létrehozásakor még végrehajtandó kódrészletet az ***„Extra code”*** mezőben kell begépelni. A kódrészlet az *1. listában* látható. Ez a kód fogja lekérdezni a felhasználó saját könyvtárát, majd azt kiegészíti a konfigurációs állomány nevével. Ez a változó a ***homedir***, mely a program futása során mindvégig használható.

A hozzáadott kód utolsó sorában még a `load_config()` függvény segítségével betöltjük a konfigurációs állományt és feltöltjük a beolvasott elemekkel a legördülő listát.

A függvény sorai a *2. listán* tanulmányozhatók a további kiegészítő függvények mellett. A programban mindössze négy különleges szerepű eljárásra van szükség.

1. lista

```
homedir=(char *)calloc(128, sizeof(char));
homedir=getenv("HOME");
strcat(homedir, CONFIGNAME);
load_config(homedir);
```

Az `initialize()` függvény a program indulásakor memóriát foglal a listaelemek számára. A `cleanup()` függvény tulajdonképpen ennek a memóriaterületnek a felszabadítására szolgál továbbá a program befejeződése előtt felszabadítja a `homedir` változó számára foglalt memóriát is. Mivel a programból többféleképpen is ki lehet lépni, helyesnek láttam a memória felszabadítására szolgáló kódrészeket külön függvényben megvalósítani.

A listában látható másik két függvény a konfigurációs állomány betöltésére és mentésére szolgál. A mentésre csak abban az esetben van szükség, ha a listában még nem szerepel a felhasználó által megadott program neve.

A lista elején még néhány állandót határozunk meg, melyek megkönnyítik a program további bővítését. Kezdetben megelégedtem maximálisan tíz listaelem tárolásával, hiszen ezzel a programmal általában a gyakran használt alkalmazásokat fogom futtatni.

A legördülő lista megvalósítására az `Fl_Choice` objektumot használjuk majd, a kézi adatbevitelhez pedig az `Fl_Input` vezérlőelemet.

Miután elhelyeztük a vezérlőelemeket az ablak területén, ideje lesz a működést is megvalósítani a különféle ***visszahívó (callback)*** függvények segítségével. Mindössze a három gombra kell meghatározni a függvényeket. Az egyes elemekre duplán kattintva előbukkan a beállításokat segítő párbeszédablak, melyben minden elemnek érdemes egy beszédes nevet adni. A visszahívó függvények megvalósítására is ebben a párbeszédablakban van lehetőségünk.

Kezdjük tehát a konfigurációs állomány alapállapotba hozásával. A függvény rendkívül egyszerű, mindössze egy kérdező ablakban (`fl_ask()` függvény) megerősítést kérünk a felhasználótól, majd annak pozitív válasza után töröljük a konfigurációs állományt és a listaelemeket. Az állomány törléséből nem származhat semmilyen hátrányunk, hiszen amennyiben a program indulásakor nem található ilyen állomány, a program folytatja futását és üres listából választhatunk. A felhasználó által megadott alkalmazást kilépéskor a beállításokat tároló állományba írjuk, így újra létrehozzuk és tároljuk a beállításokat. Az állomány törlésére az `unlink()` rutint használjuk, tehát a programunk elején szükséges az `#include <unistd.h>` sor megadása. A listaelemek törlésére az `Fl_Choice` objektum `remove()` függvényét használjuk egy ciklusban végrehajtva.

A megadott vagy kiválasztott alkalmazás indítására a ***„Run program"*** gomb visszahívó függvényében lesz lehetőségünk. A függvény egy lehetséges megvalósítását a *3. lista* mutatja.

A függvényben elsősorban megvizsgáljuk, hogy kézi adatbevitel történt-e vagy a felhasználó a listából választott valamilyen programot. Az új alkalmazás megadásakor

2. lista

```
#define MAX_ELEMS 10
#define CONFIGNAME "/.flrun.cfg"
static char *config[MAX_ELEMS];
static char *homedir;

void save_config() {
 int i;
 FILE *ofile;
 ofile=fopen(homedir, "w+");
 if (!ofile) {
   fprintf(stderr, "cannot open output file\n");
   ➥return;
 }
 for (i=0; i<proglist->size(); i++) {
   if (proglist->size()) {
      Fl_String str=proglist->text(i);
      fprintf(ofile, "%s\n", (char*)str);
   }
 }
 fclose(ofile);
 return;
}

void load_config(const char* confname) {
 int i;
 FILE *infile;
 if (confname==NULL) {
   fprintf(stderr,
      "config filename is invalid\n");
 return;
 }
 infile=fopen(confname,"r");
 if (!infile) {
 fprintf(stderr, "configfile not found\n");
 ➥return;
 }
 for (i=0; i<MAX_ELEMS-1; i++) {
 if (feof(infile)) break;
 fscanf(infile, "%s\n", config[i]);
 if (config[i]) proglist->add(config[i]);
 }
 fclose(infile);
 return;
}

void initialize() {
 int i;
 for (i=0; i<MAX_ELEMS-1; i++)
   config[i]=(char*)calloc(128, sizeof(char));
}

void cleanup() {
 int i;
 for (i=0; i<MAX_ELEMS-1; i++)
   if (config[i]) free(config[i]);
 free(homedir);
}
```

3. lista

```
static void cb_btOK(Fl_Return_Button*, void*) {
 char *mstr=(char *)calloc(128, sizeof(char));
 strcpy(mstr, proginput->value());
 strcat(mstr, (const char*)"&");

 // Nem adtunk meg kézzel semmit
 if (strncmp(mstr, "&", 1)) {
   system(mstr);
   free(mstr);
   mainWindow->hide();
   // Ha meg nincs benne a listaban a program,
   // akkor fel kell vennünk
   if (proglist->size()<10) {
      if (!proglist->find(proginput->value())) {
         proglist->add(proginput->value());
         // mentjuk a listat
         save_config();
      }
   }
   return;
 }
 if (proglist->size()>0) {
   Fl_String str=proglist->text(proglist->
   ➥value())+"&";
   if (!str) return;
   system(str.c_str());
 }
 mainWindow->hide();
}
```

a `system()` hívással végrehajtjuk a megadott psrogramot és megvizsgáljuk a listaelemeket. Ha nem találjuk a legördülő lista elemei között az alkalmazás nevét, hozzáadjuk a meglévő elemekhez és a `save_config()` függvény meghívásával tároljuk a beállításokat.
Ha a listából választott a felhasználó, akkor nincs más tennivaló, mint kiolvasni az elem szövegét és szintén a `system()` hívással végrehajtani a programot. Mindkét esetben az ablakot elrejtjük a `hide()` metódusának segítségével, hiszen az ***eFltk*** fő ciklusa addig fut, amíg valamilyen programablak látható a képernyőn. A program tehát befejezi működését a gomb megnyomása után.
A harmadik visszahívó függvényben egészen egyszerűen csak elrejtjük a fő ablakot, így a ***„Cancel"*** gomb használata szintén a program befejeződését eredményezi.
Érdemes még néhány szót ejteni a főprogramról is. Az ***eFluid***-ban a ***New->Code->function/method*** menüpontok választásával hozhatjuk létre a `main()` függvényt. A megjelenő párbeszédablakban minden beviteli mező tartalmát töröljük ki, hogy az ***eFluid*** tudtára adjuk, hogy a `main()` függvény kódját szeretnénk megadni. A főprogram kódja mindössze a 4. listában látható néhány sorból áll.

4. lista

```
int main (int argc, char **argv) {

 initialize();
 mainWindow=make_window();
 mainWindow->show();
 Fl::run();
 cleanup();
 return 0;
}
```

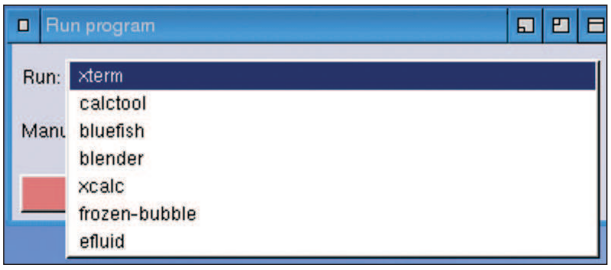
A függvényben meghívjuk a memóriaterületek lefoglalására szolgáló `initialize()` függvényt, majd létrehozzuk és megjelenítjük a program ablakát. Ezután már a szokásos `Fl::run()` függvény, vagyis az eseménykezelő ciklus futtatása következik. A ciklus befejeződése egyben az alkalmazás futásának végét is jelenti, azonban a kilépés előtt a `cleanup()` függvény segítségével felszabadítjuk a lefoglalt memóriaterületeket.
A fenti függvények megvalósítása után már nincs más tennivalónk, mint menteni a grafikus felületet és vele együtt a programot is. Az ***eFluid***-dal készült alkalmazások kiterjesztését érdemes ***.fl***-ként megadni, mert a fejlesztőeszköz is ilyen kiterjesztésű állományokat tekint sajátjának és alapállapotban ezeket jeleníti meg az állományok megnyitását segítő párbeszédablakban is.
A mentést később billentyűk segítségével is elvégezhetjük, mégpedig a ***CTRL+S*** kombináció alkalmazásával.
Ne feledkezzünk meg a forrásállományok elkészítéséről sem. Használjuk a ***CTRL+W*** billentyűket, így az ***eFluid*** elkészíti a teljes programunkat tartalmazó forráskódot és hozzáfoghatunk a program fordításához. Ennek legegyszerűbb módja, ha egy terminál ablakban kiadjuk a következő parancsot:

```
efltk-config ---compile runapp.cpp
```

Ebben az esetben a grafikus felületet ***runapp.fl*** néven mentettem és így a generált programkódot tartalmazó állomány neve is megegyezik a mentés során megadottakkal. A parancs hatására létrejön a futtatható állomány, amit érdemes megszabadítani a felesleges dolgoktól.
Adjuk ki a

```
strip -s ./runapp
```

parancsot - a program könyvtárában -, hogy a futtatható állományt megszabadítsuk a szerkesztés során létrejött szimbólumoktól. A program futását ez a művelet nem befolyásolja, viszont jelentősen lecsökkenthetjük a futtatható állomány méretét.
Amennyiben összetett programrendszert készítünk érdemes megismerkedni a kézi fordítás mikéntjével is. Amikor az ***eFltk*** segédprogramját használjuk látható, hogy milyen kapcsolókra van szükség a fordításhoz. Ezeket a kapcsolókat kiegészíthetjük tetszőleges ***GCC*** opciókkal vagy készíthetünk egyszerű ***Makefile***-t is a fordítás megkönnyítéséhez.



2. ábra Alkalmazás kiválasztása

Ebben a példában néhány kiegészítő opciót is megadtam a fordítónak, majd az alábbi parancssor segítségével, kézzel fordítottam le a programot:

```
gcc -o runapp -O3 -funroll-loops -fexceptions
➥-march=pentiumpro -DFL_SHARED
    -I/usr/X11R6/include -L/usr/X11R6/lib -lefltk
    ➥-lX11 -lXext -lm runapp.cpp
```

A program elkészítése után ne felejtsünk el biztonsági másolatot létrehozni az állományokról.
Végül nem maradt más hátra, mint az *IceWM* tudtára adni, hogy mostantól programok futtatására ezt a programot szeretnénk használni. Nyissuk meg az *IceWM* beállításait tartalmazó *~/.icewm/preferences* állományt (amennyiben nem találunk ilyen állományt, az *IceWM* adatait tartalmazó könyvtárból át tudjuk másolni az alapértelmezett beállításokat), és keressük meg benne a `RunCommand=` kezdetű sort. Itt kell megadnunk a futtatható állomány elérési útját, majd az *IceWM* újraindítása után máris használatba vehetjük ezt a hasznos segédprogramot. A *2. ábrán* látható az elkészült program működés közben.
Ezzel végére ért az *eFltk* bemutatását célzó írások sora. Természetesen nem adhatok az adott keretek között teljes referenciát minden osztályról, de remélhetőleg sikerült ízelítőt adnom a grafikus elemkészlet lehetőségeiből és a vállalkozó kedvű, tanulni vágyó olvasók folytatják a készlet felfedezését és használatát. Ehhez nagy segítség lehet a továbbiakban az *Fltk* dokumentációja, hiszen nem minden objektumról találunk leírást az *eFltk*-ban. Az eredeti *Fltk* dokumentáció azonban részletesen, példákkal megvilágítva bemutatja az elemkészlet minden lehetőségét, legyen szó szövegszerkesztő készítéséről vagy *OpenGL* alapú három dimenziós megjelenítésről.
A készlethez az interneten is találhatunk kiegészítő vezérlőket és számos példán keresztül elmélyedhetünk használatának rejtelmeiben. Izgalmas felfedező utat kívánva egy kis időre búcsúzom kedves olvasóimtól, jó nyaralást és tanulást kívánok mindenkinek.

Fábián Zoltán (dzooli@freemail.hu)
26 éves, jelenleg oktatóként dolgozik, szabadidejében szívesen foglalkozik Blenderrel, programozással és elektronikai tervezéssel. Szereti a természetet, túrázást, úszást és a kellemes baráti társaságot.