Itt az idő, hogy mindent újragondolj!

Az EFL alapjai

Az EFL egy függvénykönyvtár, és hasonlóan a Gnome, és a KDE keretrendszereihez, ez az Enlightenment DR17 (E17) alapja. Minek még egy, hiszen már van nekünk GTK, QT, és még sorolhatnánk a grafikus eszköztárakat?! Ez sem lesz szebb, gyorsabb, vagy egyszerűbb! Nos, aki így vélekedik, annak itt az idő, hogy mindent újragondoljon.

„*Itt az idő, hogy mindent újragondolj!*" Ez ugyanis a szlogen. És valóban, az *E17*-et készítő csapatnak ez sikerült is. Az *EFL* tagjai úgy lettek tervezve, hogy a lehető legkevesebb erőforrást használják fel számítógépünkből, egyszerűen kezelhetőek legyenek mind programozói, mind felhasználói szemszögből, és nem utolsó sorban elálljon a lélegzetünk az grafikus effekteknek köszönhetően. Jó, jó, hiszen mindenki ezt mondja a saját programjáról! Nos, az a helyzet, hogy a csapat egyik tagja egy 100 Mhz-es *Pentium I*-esen teszteli rendszeresen az alkalmazásokat, elég gyorsak-e azon is, és meg van elégedve az eredménnyel.
Mik is alkotják hát az *EFL*-t? Kezdjük az alapokkal.

Evas

Az *Evas* egy nagyon jól portolható és okos képernyő-leképező. Gyorsaságát főleg neki köszönheti az egész *EFL*, hiszen ezen alapul. Nem csak *X* szerveren használhatjuk, hiszen fut *Framebufferen*, *DirectFB*-n, *OpenGL*-lel is gyorsíthatjuk működését, sőt „okostelefonokra" és *PDA*-kra is létezik. Ellát minket jó minőségű képátméretezéshez, élsimított (anti-aliased) betűk létrehozásához, színátmenetek készítéséhez, egyszerű vonalak előállításához való eszközökkel, és ez még nem minden. Az embernek amiatt sem kell törnie a fejét, hogy hány színt képes kezelni az *X* szerverünk. Elég okos hozzá, hogy megtalálja a maximálisan megjeleníthető színek számát, így akár egy fekete-fehér kijelzőn is fut. A megfelelő *Doxygen* dokumentáció a ➲ *http://enlightenment.sourceforge.net/doxy/evas/* címen érhető el.

Ecore

Ez a könyvtár felelős az eseménykezelésekért, a húzd-és-ejtsd (*drag-and-drop*) rendszerért, a kijelölések kezeléséért, és az alapszintű rendszerekért. Nagyon jól bővíthető, és gyors. Használhatjuk alapjául az *Evast*, de közvetlen *Framebufferen* és *X* szerveren is fut. Tartalmaz modulokat szövegkonverzióra, folyamatok közti kommunikációra, kliens-kiszolgáló kapcsolatokra, és még sok minden másra is.

Embryo

Az *Embryo* egy *C* szerű szintaxissal rendelkező szkriptnyelv. Minden *Embryo* program egy védett környezetben (*sandbox*) fut, ami meggátolja abban, hogy rendszerünkben kárt tegyen, viszont ezáltal nem alkalmazható olyan feladatokra, mint amilyeneket *Bash*, *Perl*, vagy *Python* programokkal oldanánk meg. Ebből a védett környezetből nem tud fájlokat kezelni, memóriaterületet foglalni, más folyamatokat elérni, és a hálózathoz csatlakozni sem. Az *Embryo* kód egy nagyon kicsi, és gyors virtuális gépben fut. Az egész virtuális gép kódja nem több, mint háromezer sor, ami szép teljesítmény.

Edje

Az *Edje* egy olyan könyvtár, ami megadja a lehetőséget, hogy a grafikus felhasználói felületet elválasszuk

a tényleges munkát végző kódtól. Ami különlegessé teszi, az az a tulajdonsága, hogy mi magunk csinálhatjuk meg a grafikus elemeket, amiket később elláthatunk olyan *Embryo* kódokkal, amik azt animálják, vagy éppen átalakítják a képeket. Így készíthetünk akár *Flash* animáció szerű programokat is. Az *Embryo* forrásfájlokat a képekkel együtt *EET* kiterjesztésű állományokban tartja.

EET, EDB

A két rövidítés egy-egy állománytípust jelöl. Az ilyen fajta fájlok bináris formában tartalmaznak kulcsérték párokat. Az *EET zip*-szerű, tehát tömörítheti is a bevitt adatot, és inkább fájlok tárolására való, ám egy *zip*-nél jóval egyszerűbb felépítésű.
Az *EDB* fájlok beállításokat tárolnak. A kulcs-érték párokhoz kapcsolódik még egy típus is, ami lehet `int` (egész), `str` (karakterlánc), `float` (lebegőpontos szám), vagy `data` (bináris adat).
Az *EET Doxygen*-je a ➲ *http://enlightenment.sourceforge.net/doxy/eet/* címen érhető el, az *EDB*-nek viszont nincs ilyenje, ezért a jó minőségű, megjegyzésekkel tűzdelt fejállományt kell használnunk.

Epeg, Epsilon

Két képkicsinyítőről, villámnézet (*thumbnail*) készítőről van szó. Az *Epeg* (a készítők szerint) a világ leggyorsabb *JPEG* kicsinyítő könyvtára. Az *Epsilon* képes saját maga kezelni PNG formátumú képeket is, és az Imlib2 könyvtárral együttműködve ez a sor még kiegészül az XCF és TIFF formátumokkal is. Az Epeggel kombinálva fantasztikus sebességgel készít villámnézetet még a nagy felbontású képekből is.
Az *Epeg Doxygen* dokumentációja a ➲ *http://enlightenment.sourceforge.net/doxy/epeg/* oldalon érhető el, az *Epsiloné* pedig ugyanitt de a /epsilon könyvtár alatt.

Etox

Az *Etox* nagyon hasznos, ha szövegek formázásáról van szó. Képes mindenféle effektre, mint szövegeket színezése, képek körülfolyatása, átméretezése, rétegekbe rendezése, vagy akár mozgatása. Mindezt egy sima ablakban, szövegszerkesztő nélkül. Szintén nincs *Doxygenje*, de a fejállományok itt is segítségünkre vannak.

Emotion

Az *Emotion* mozgóképek, videók kezelésére való. Alapjául a *libxine*, vagy a *Gstreamer* használható, így minden formátumot képes lejátszani, amit azok is. Segítségével akár 18 sorban írhatunk egy *DVD* lejátszót. Nincs *Doxygen* dokumentációja, a fejállományokat kell tanulmányoznunk.

EWL

Ez az *EFL* grafikus eszköztára. Bár létezik egy *ETK* nevű hasonló funkciójú könyvtár is, arról egy szót sem ejt az *Enlightenment* hivatalos oldala. Elég nehézkes a *Doxygen* böngészése, sok mindent nehéz benne megtalálni, ezért ajánlom a közvetlen fejállományokat tanulmányozni. A *Doxygen* elérhető a ➲ *http://enlightenment.sourceforge.net/doxy/ewl/ címen.*

Lássuk mindezt a gyakorlatban!

Most, hogy megismerkedtünk az *EFL* építőelemeivel, lássuk, hogyan is kell használni őket. A következő példákban az *EWL*, *EDB*, és *EET* könyvtárak lesznek a főszereplők. A programokat a

```
gcc -o program prog.c
➥`ewl-config -cflags -libs`
➥`eet-config -cflags -libs`
➥`edb-config -cflags -libs`
```

paranccsal fordíthatjuk le.

1. Lista hello.c

```
 1: #include <Ewl.h>
 2: #include <Eet.h>
 3: #include <Edb.h>
 4: #include <stdio.h>
 5:
 6: void vege__ (Ewl_Widget *hivo, void
    ➥*esemeny, void *extra) {
 7: ewl_widget_destroy(hivo);
 8: ewl_main_quit();
 9: }
10:
11: void eet__ (Ewl_Widget *hivo, void *esemeny,
    ➥void *extra) {
12: char *fajlnev;
13: char *szoveg;
14: Eet_File *fajl;
15: Ewl_Widget *bevitel;
16:
17: bevitel = (Ewl_Widget *)extra;
18: fajlnev = ewl_text_text_get
    ➥(EWL_TEXT(bevitel));
19:
20: fajl = eet_open(fajlnev,
    ➥EET_FILE_MODE_WRITE);
21: if(!fajl) return;
22:
23: szoveg = "Ez egy mondat, amit eet-vel írtunk
    ➥ki.";
24: eet_write(fajl, "eleresi_kulcs", szoveg,
    ➥(strlen(szoveg)+1), 1);
25: eet_close(fajl);
26: }
27:
28: void edb__ (Ewl_Widget *hivo, void *esemeny,
    ➥void *extra) {
29: char *fajlnev;
30: char *szoveg;
31: E_DB_File *fajl;
32: Ewl_Widget *bevitel;
33:
34: bevitel = (Ewl_Widget *)extra;
35: fajlnev = ewl_text_text_get
    ➥(EWL_TEXT(bevitel));
36:
```

1. Lista folytatás

```
37: fajl = e_db_open(fajlnev);
38: if(!fajl) return;
39: e_db_property_set(fajl, "program", "Hello");
40: e_db_str_set(fajl, "/eleresi/kulcs", "Ez egy
    ➥mondat, edb-ben tárolva");
41: e_db_close(fajl);
42: }
43:
44: int main (int argc, char **argv) {
45: Ewl_Widget *ablak, *vbox, *hbox,
46:       *cimke, *bevitel,
47:       *gomb_eet, *gomb_edb;
48:
49:
50: if(!ewl_init(&argc, argv)) return 1;
51:
52: /** Az ablak **/
53: ablak = ewl_window_new();
54: ewl_window_title_set(EWL_WINDOW(ablak),
    ➥"Hello, mi?");
55: ewl_window_name_set(EWL_WINDOW(ablak),
    ➥"EwlDemo");
56: ewl_window_class_set(EWL_WINDOW(ablak),
    ➥"EwlDemo");
57: ewl_object_size_request(EWL_OBJECT(ablak),
    ➥300, 50);
58: ewl_callback_append(ablak,
    ➥EWL_CALLBACK_DELETE_WINDOW, vege__,NULL);
59:
60: /** A vbox tároló **/
61: vbox = ewl_vbox_new();
62:
63: /** A hbox tároló **/
64: hbox = ewl_hbox_new();
65:
66: /** A címke **/
67: cimke = ewl_text_new();
68: ewl_text_text_set(EWL_TEXT(cimke), "A fájl
    ➥elérési útja:");
69: ewl_object_alignment_set(EWL_OBJECT(cimke),
    ➥EWL_FLAG_ALIGN_RIGHT);
70:
71: /** A beviteli mező **/
72: bevitel = ewl_entry_new();
73:
74: /** Az eet-be író gomb **/
75: gomb_eet = ewl_button_new();
76: ewl_button_label_set(EWL_BUTTON(gomb_eet),
    ➥"Eet-be írás");
77: ewl_callback_append(gomb_eet,
    ➥EWL_CALLBACK_CLICKED, eet__, bevitel);
78:
79: /** Az edb-be író gomb **/
80: gomb_edb = ewl_button_new();
81: ewl_button_label_set(EWL_BUTTON(gomb_edb),
    ➥"Edb-be írás");
82: ewl_callback_append(gomb_edb,
    ➥EWL_CALLBACK_CLICKED, edb__, bevitel);
83:
84: /** Elemek hozzáadása a tárolókhoz **/
85: ewl_container_child_append
    ➥(EWL_CONTAINER(ablak), vbox);
86: ewl_container_child_append
    ➥(EWL_CONTAINER(vbox), cimke);
87: ewl_container_child_append
    ➥(EWL_CONTAINER(vbox), bevitel);
88: ewl_container_child_append
    ➥(EWL_CONTAINER(vbox), hbox);
89: ewl_container_child_append
    ➥(EWL_CONTAINER(hbox), gomb_eet);
90: ewl_container_child_append
    ➥(EWL_CONTAINER(hbox), gomb_edb);
91:
92: /** Elemek megjelenítése **/
93: ewl_widget_show(ablak);
94: ewl_widget_show(vbox);
95: ewl_widget_show(hbox);
96: ewl_widget_show(cimke);
97: ewl_widget_show(bevitel);
98: ewl_widget_show(gomb_eet);
99: ewl_widget_show(gomb_edb);
100:
101:    ewl_main();
102:    return 0;
103: }
```

1-5. sorban betöltjük a szükséges fejállományokat. Az ***Ewl.h***, ***Eet.h***, ***Edb.h*** fájlok értelemszerűen az ***EWL***, ***Eet***, és ***EDB*** könyvtárakhoz kellenek.
A 7-43. sorig a visszahívásokat definiáljuk, ezekre később térek ki.
A `main` függvény a 45. sorban kezdődik. A paraméterekre az ***EWL*** inicializálásakor van szükségünk, ugyanis van néhány opció, amiket a könyvtár maga kezel le. Ezek az opciók a téma beállításához, a hardveres ***OpenGL***, vagy szoftveres ***X11*** leképezés megválasztásához használhatók. Ez az inicializálás az 51. sorban történik meg.
Az 55. sorban hozzuk létre az ablakunkat. A függvény `Ewl_Widget *` típussal tér vissza. Ez azért van, mert egy objektumnak vannak saját, és általános tulajdonságai. Az általános tulajdonságokat minden egyes objektumnál ugyanazokkal a függvényekkel állítjuk be, így azok ugyanolyan típusú mutatókat várnak paraméterül. Az átalakítást a fejállományokban definiált állandók végzik. Általános tulajdonságnak hívhatjuk az `ewl_object` és `ewl_callback` kezdetű függvényeket.
Az objektum saját tulajdonságait csak egy adott típusra állíthatjuk be. Például egy beviteli mezőre nem állíthatunk be ablakosztályt, ellenkező esetben az alkalmazásunk futtatáskor hibával áll le. Ablakunkhoz tartozó saját tulajdonságokat az 55-57. sorokban

állítjuk be. Az `ewl_window_title_set` értelemszerűen az ablak címét állítja be, esetünkben `Helló, mi?`-re. Az 56-57. sorokban beállított tulajdonságok az ablakkezelőnek szükségesek, például az ***E17*** ez alapján társítja az ikonokat futás közben az alkalmazásokhoz. Az 58. sorban beállítjuk az ablak méretét. Ezt a függvényt bármilyen objektumra meghívhatjuk, ahogy az 59. sorban található hívást is, ami visszahívást rendel az ablakunkhoz. Láthatjuk, hogy ehhez a híváshoz nem kell átalakítanunk a mutató típusát. Az `ewl_callback_append` függvény második paramétere a visszahívás típusa, jelen esetben az ablak bezárása. A harmadik kapcsoló a meghívandó függvény, a negyedik pedig egy extra objektum, amit átadhatunk. Esetünkben erre nincs szükség, így `NULL`.
Így el is érkeztünk a visszahívásokhoz, ugorjunk tehát a 7. sorra, a `vege__` függvény kezdetéhez. Egy visszahívás három változót vár paraméterül: az első, `Ewl_Widget` típusú maga a hívó. A második az esemény típusa, a harmadik pedig a fent említett „extra”. Minden visszahívás `void` típusú, tehát nem tér vissza értékkel.
A `vege__` függvény `ewl_widget_destroy` hívása az ablak, és összes gyermeke megsemmisítéséről gondoskodik. Az `ewl_main_quit` hívás az ***EWL*** megjelenítési ciklusát állítja le. Ezt a ciklust az `ewl_main` hívással indítjuk majd el.
A 62. és 65. sorban létrehozott `vbox` és `hbox` tárolókhoz adott elemek

1. ábra A futó program

függőlegesen, illetve vízszintesen egymás alatt, illetve mellett lesznek. Használhatnánk bonyolultabb elrendezési mechanizmust alkalmazó tárolókat is, de ezek céljainknak tökéletesen megfelelnek.
Az ***EWL*** `Text` típusa egy nem írható szövegmezőt, címkét jelöl. Ezt hozzuk létre a 68. sorban. A címke szövegét a következő sorban állítjuk be, de azt akár az `ewl_text_new` függvénynek átadott mutatóval is megtehetnénk.
Az `ewl_object_alignment_set` hívás szintén minden típusra alkalmazható, és az objektum elhelyezkedését szabja meg, az átadott állandó segítségével. A mi esetünkben az ablak jobb oldalára igazítja a widgetet.
A 73. sorban inicializált `Entry` típusú objektum egy beviteli mezőt jelöl. Vele egyelőre nincs más dolgunk.
A 76. és 81. sorban két gombot hozunk létre, amik szinte ikrei egymásnak, csak nevükben, feliratukban térnek el. A gombok címkéjét nem `Text` típusúra átalakítva kezeljük, mint a beviteli mezőket, hanem egy saját függvény segítségével.
A gombok visszahívásai, az `eet__` és `edb__` függvények a gombokra való kattintáskor hívódnak meg, és ***eet*** illetve ***edb*** adatformátumban kiírnak egy mondatot a bevitel objektumba beírt fájlnevet felhasználva.
Akkor tehát vizsgáljuk meg az `eet__` függvényt. Az első négy sorban a szükséges változókat deklaráljuk, majd a következő sorban az extra változót típuskényszerítéssel átalakítjuk `Ewl_Widget`-re, így a változó típusát a 19. sorban tovább tudjuk kényszeríteni `Ewl_Text`-re, ezt végzi igazából az `EWL_TEXT` (bevitel) kifejezés.
A 21. sorban nyitjuk meg a beviteli mezőbe írt névhez tartozó fájlt, majd rögtön az utána lévő sorban leellenőrizzük, hogy ez a művelet sikerült-e.
A 25. sorban kiírjuk a mondatunkat: az második paraméter az elérési kulcs, amivel később mondatunkra hivatkozni tudunk majd, a harmadik maga a tárolni kívánt adat. A negyedik kapcsoló a tárolni kívánt adat hossza, esetünkben a karakterláncé, beleértve a végét jelző 0 karaktert is. Az utolsó várt kapcsoló 0 vagy 1 értéket kaphat aszerint, hogy szeretnénk-e, hogy adatunk tömörítve legyen. Mi természetesen akarjuk! A 26. sorban végül lezárjuk eet fájlunkat.
Az `edb__` függvényben az egyetlen új dolog az ***edb*** fájlok kezelése.
A 38. sorban megnyitjuk a fájlt mindenféle műveletre.
Az `e_db_open_mode` függvénnyel meghatározhatnánk a megnyitás módját az `open` függvényhez hasonlóan, az `O_` állandókkal.
A 39. sorban egy „tulajdonságot” állítunk be. Ez a mező nem szerepel a többi között, ha lekérjük a kulcsok listáját, és nem elérhető egy egyszerű, valamilyen `get` hívással. Mondhatni „rejtett”, de mégsem. Arra használható, hogy az alkalmazások megtudják: miféle adatbázissal van dolguk.
Tudni kell, hogy nem csak egy ilyen állítható be, ilyen módon sok mindent tárolhatunk.
A következő sorban viszont egy „igazi” mezőt készítünk. Az elérési kulcs, mint láthatjuk, némileg más formátumú, mint az ***eet*** esetében, azonban ez nem az adatformátumtól függ. Az ***E17***, és az ***EFL***-t használó programok inkább ez utóbbi formát használják, így egyszerűen csoportosíthatóak a beállítások, tehát érdemes ez utóbbira szavazni, bár ízlés szerint bármilyen kulcs használható.

2. Lista eet_olvas.c

```
 1: #include <Edb.h>
 2: #include <stdio.h>
 3:
 4: int main (int argc, char
    ➥**argv) {
 5: int kulcs_sz, i;
 6: char **kulcsok;
 7: E_DB_File *fajl;
 8:
 9: if(argc != 2) { printf("Használat:
    ➥\n\t%s [fáljnév]\n", argv[0]);
10: return 1; }
11:
12: fajl = e_db_open_read(argv[1]);
13: if(!fajl) { printf("Nem tudtam megnyitni
    ➥a fájlt"); return 1; }
14: kulcsok = e_db_match_keys(fajl,"*",
    ➥&kulcs_sz);
15:
16: for (i = 0; i < kulcs_sz; i++) {
17: printf("típus: %s ", e_db_type_get(fajl,
    ➥kulcsok[i]));
18: printf("| %s : \"%s\"\n", kulcsok[i],
    ➥e_db_str_get(fajl,kulcsok[i]));
19: }
20:
21: e_db_close(fajl);
22: free(kulcsok);
23: return 0;
24: }
```

3. Lista edb_olvas.c

```
 1: #include <Edb.h>
 2: #include <stdio.h>
 3:
 4: int main (int argc, char **argv) {
 5: int kulcs_sz, i;
 6: char **kulcsok;
 7: E_DB_File *fajl;
 8:
 9: if(argc != 2) { printf("Használat:\n\t%s
    ➥[fáljnév]\n", argv[0]);
10: return 1; }
11:
12: fajl = e_db_open_read(argv[1]);
13: if(!fajl) { printf("Nem tudtam megnyitni
    ➥a fájlt"); return 1; }
14: kulcsok = e_db_match_keys(fajl,"*",
    ➥&kulcs_sz);
15:
16: for (i = 0; i < kulcs_sz;
    ➥i++) {
17: printf("típus: %s ", e_db_type_get(fajl,
    ➥kulcsok[i]));
18: printf("| %s : \"%s\"\n",
    ➥kulcsok[i],
    ➥e_db_str_get(fajl,kulcsok[i]));
19: }
20:
21: e_db_close(fajl);
22: free(kulcsok);
23: return 0;
24: }
```

A 85. sortól a tárolókhoz adjuk a widgeteket, majd a 93. sortól megjelenítjük őket. Figyeljük meg, hogy az objektumok sorrendje futáskor nem a megjelenítési sorrenden múlik, hanem a tárolókhoz adáskor határozzuk meg azt.
Végül a 101. sorban elindítjuk az ***EWL*** megjelenítési ciklusát, amit majd az ablak lezárásakor léptetünk ki, hogy programunk befejeződjön.

Eet fájlok olvasása

Az ***Eet*** és ***Edb*** könyvtárak telepítésekor megkapjuk az ilyen típusú fájlok kezeléséhez szükséges programokat is, ***eet***, és `edb_ed` parancsok formájában. Az ***Eet*** fájlok, mint említettem inkább más állományok tárolására szolgálnak, hasonlóan egy ***zip*** tömörítésű fájlhoz, ezért az ***eet*** program a kulcsokat fájlnevekként kezeli, tartalmukat nem a képernyőre írja, hanem az kulcs által meghatározott fájlba. Ennek kiküszöbölésére, és annak szemléltetésére, hogy hogyan lehet ***C*** programból ezeket a fájlokat kezelni, íme egy újabb példa (*2. Lista*).
A 12. sorban megnyitjuk a programnak átadott fájlt, majd a 14. sorban lekérjük az elérhető kulcsok listáját. A második paraméter egy szabályos kifejezés, amivel meghatározzuk, hogy mi alapján akarjuk szűrni az eredményül kapott kulcsokat. Jelen esetben nincs ilyenről szó, így megteszi minden kulcs.
A 16-18. sorban végigmegyünk a kulcsokon, és egyenként kiírjuk őket

```
kulcs : "érték"
```

formában. Az értékeket az `eet_read` függvénnyel kapjuk meg. Harmadik paraméternek egy mutatót vesz, amibe beleírja a visszatérési adat méretét. Most erre nincs szükségünk, így nyugodtan lehet `NULL`.
A 22. sorban, ahogy azt illik, felszabadítjuk a kulcsokat tartalmazó tömb által lefoglalt memóriát, és a 23. sorban befejezzük a program futását. Az ***Eet Doxygen*** külön felhívja a figyelmet, hogy egyenként a tömbben lévő mutatókat ne szabadítsuk fel!

Edb fájlok olvasása

Mint említettem, létezik egy `edb_ed` nevű program, amivel az *edb* fájlokat kezelni tudjuk. A létező kulcsok listáját, és azok típusát például az

```
edb_ed valami.db match "*"
```

paranccsal kérhetjük le, ahol a zárójelek között egy szabályos kifejezést adhatunk meg.
Ha viszont egy programból szeretnénk ezeket a fájlokat kezelni, egy hasonló kódra lesz szükségünk (*3. Lista*). Csak olvasására megnyitni egy fájlt az `e_db_open_read` függvénnyel tudunk, ahogy azt láthatjuk a 11. sorban.
A 14. sorban lévő `e_db_match_keys` szintén egy szabályos kifejezést, és egy egész mutatót vár, előbbit a kívánt kulcsok szűrésére, utóbbit a visszaadott kulcsok számának tárolására használja.
A 17. sorban lekérjük a kulcshoz tartozó típust, és kiírjuk a képernyőre. A 18. sorban aztán karakterlánc típusúként kérjük le az adatot, és a kulccsal együtt kiírjuk a képernyőre. Így a teljes kimenet elemenként így néz majd ki:

```
típus: str | kulcs : "érték"
```

A 21. sorban lezárjuk a fájlt, utána felszabadítjuk a kulcsokat tartalmazó tömb által lefoglalt memóriaterületet, és befejezzük a programunkat.

Összegzés

Az *E17* és az *EFL* úgy lettek tervezve, hogy ha bármire szükség van a fejlődéshez, egyszerűen lehessen őket bővíteni. Jelenleg mindkét komponens aktív fejlesztés alatt áll, így folyamatosan változik.
Az *Enlightenment* hivatalos oldalán (➲ *http://enlightenment.org*) lévő könyvek már mind elavultak, így csak a *Doxygenre*, és a fejállományokra bízhatjuk magunkat.
Jelenleg nincsen stabil verzió egyik *EFL* könyvtárból sem, így a *CVS* fából kell őket kinyernünk, és lefordítanunk, de létezik néhány gyakran frissülő hely, ahol bináris csomagokat érhetünk el Debian rendszerekhez. A *Gentoo Linux Portage* fájában alapból benne vannak az *EFL* csomagok, telepítési útmutató ért keressük fel a ➲ *http://gentoo-wiki.org*-ot.
Található továbbá egy nem-hivatalos tár, ahová éjjelente kerülnek fel a napi *CVS* lenyomatok *.tar.gz* formátumban, így nem kell beleásnunk magunkat a *CVS* kezelésébe, és egyszerűen nekiállhatunk a fordításnak. A komponenseket a következő sorrendben kell lefordítanunk:

eet, edb, evas, ecore, embryo, imlib2, edje, e(vagy enlightenment), epeg, epsilon, esmart, entrance, ewl

Azt hiszem ennyi kiindulásnak elég volt, talán még sok is. Akinek van kedve fejleszteni programokat az *EFL*-re, tegye bátran, a csapat szívesen lát mindenkit!

Párkányi Péter
(peter.parkanyi@gmail.com)
1991-ben születtem, és idén elsős vagyok egy pécsi gimnáziumban. Szeretem a jó zenét, szeretek kerékpározni, és kosárlabdát is szívesen játszom. Jelenleg Gentoo Linuxot használok.