Programozás Alapjai 1
NHF programozói dokumentáció

Péter Bertalan Zoltán

2017. december 4.

drunt

1 Státusz

Sajnos nem mondható el a programról a hibátlan működés minden szempontból, egyelőre. Ellenben a program fő funkciói megfelelően működnek. Sőt, tulajdonképpen minden funkció megfelelően működik, azonban helyenként még előfordulhatnak hibák, hiányosságok. A grafikus kezelőfelület fejlesztés alatt áll, de félig-meddig már használható.

1.1 Javításra szorul

- grafikus mód
- egyszeri futtatások (a `drunt` indítása mindössze egy parancs futtatásának erejéig, amit argumentumként adunk át)
- személyre szabás, konfiguráció (például alapértelmezetten megnyitott naptárfájl)
- egyéb javítani valók: valamit mindig lehet fejleszteni, szépíteni, korrigálni... A tökéletességtől még akkor is valószínűleg messze van a program, ha már tökéletesnek látszik

2 Fájlstruktúra és függvények

A kész programfájl sok-sok egyéb fájlból fordul le. Ezeket az alábbiakban leírom, tartalmukkal, fontos függvényeikkel együtt.

2.1 `drunt.c`

A fő `.c` fájl. Egyedüli feladata, hogy eldöntse, mi történjen indításkor. Ezt úgy végzi, hogy megnézi a kapott argumentumokat. Ha lehetne egyéni konfigurációt csinálni, akkor például egy `JSON` fájlban tárolt opció határozná meg, hogy milyen módban indul a program, ha nem kapott argumentumot. A program úgy íródott, hogy ezek a konfigurációk ne legyenek „hard code"-olva, tehát viszonylag kevés módosítással átírhatók az értékek. Erre nem került sor, így alapértelmezés szerint argumentum nélkül az interaktív, szövegalapú mód indul (mivel az szinte kifogástalanul működik, ellenben a grafikussal), explicit pedig a `-t` opcióval (argumentummal) választhatjuk ezt a módot. A gyerekcipőben járó GUI mód a `-g` opcióval indítható. Egyéb opciók implementálása lehetővé tenni az egyszeri futást, de jelenleg ez nincs lekódolva (annyira nem is lenne kényelmes; talán felesleges is)

Itt kerül definiálásra továbbá a globális naptár fájl és egy string, ami a legutóbb megnyitott fájl elérési útját tárolja. Oda íródik a memóriából a naptár kilépéskor.

2.1.1 `main.c`

Meghívja a naptárat inicializáló függvényt és memóriát foglal az alapértelmezett fájl elérési útnak (a sztringnek). Betölti a naptárat az alapértelmezett fájlból a memóriába, majd egy primitív `stringcompare` segítségével eldönti az argumentumok szerint, hogyan tovább. Gyorsan módosítható, de alapból argumentum nélkül interaktív, szöveges módban indul, a `-g` argumentumra pedig GUI módban.

2.2 `dbHandler.c`

Adatbázis-kezelő függvények foglalnak itt helyet. Könnyen kezelhetőnek kell lenniük, mivel többször is meghívásra kerülnek: minden alkalommal, ha új eseményt hozunk létre, törlünk, módosítunk... A következő függvényeket tartalmazza:

2.2.1 `MYERRNO ICS_load(const char* file, Calendar* cal)`

Azért felelős, hogy a `.ics` fájlokat beolvassa a memóriába, egy `Calendar` struktúrába, aminek a címe kerül átadásra, argumentumként.

A függvény először ellenőrzi a kapott fájlt (értsd: a fájl elérési útja alapján megpróbálja megnyitni a fájlt és ha sikerül, ellenőrzi (egészen pontosan először egy másik függvény kísérli meg a megnyitást, az, amelyik az ellenőrzést is végzi. Ez a függvény a `helper.c`-ben van)). Ezután nyit egy fájl stream-et és a robusztus `fgets()` függvénnyel olvassa be a fájlt, soronként. Egy buffer-be olvas és a különböző „tag"-eket figyelembe véve eldönti, mit kell csinálni. Erre egy `ICSTag` típusú tömböt készít, ami tartalmazza a különböző tag-ek szöveges

megjelenését, típusát és egy flag-et. Miután úgy érzi, egy eseményt beolvasott, beleteszi a memóriába és tovább olvas.

2.2.2 `MYERRNO ICS_write(const char* file, const Calendar* cal, WriteMode wm)`

Az előző függvény testvére, aki nem fájlból olvas be, hanem beleír. Viszonylag egyszerű a működése, mert nem kell az átalakítgatással, értelmezéssel bajlódni, mindent be lehet pakolni `fprintf()`-ekbe, formázással. Kap `WriteMode` paramétert, ami megmondja, hogy felülírhat-e már létező fájlt (ha jól emlékszem, csak így van meghívva a program során, tehát nem feltétlenül lenne szükség a funkcióra. Viszont így univerzálisabb a függvény).

2.2.3 `MYERRNO Calendar_create(Calendar* cal)`

Nem végez fájlmanipulációt, csak készít egy üres naptárat. Tulajdonképpen egy inicializáló függvény. A paraméterként megadott helyen lévő naptárat inicializálja üresre (a naptárat egy láncolt lista jelképezi: ez elkészíti a két strázsát, amik egymásra mutatnak).

2.2.4 `MYERRNO Calendar_destroy(Calendar* cal)`

Az előző függvény párja. Ez végigmegy egy naptáron (mármint a memóriában létezőn) és sorban felszabadítja az elemeit. Igazából egy egyszerű láncolt-lista-felszabadítóról van szó.

2.2.5 `MYERRNO Calendar_addVEvent(Calendar* cal, VEvent ve)`

Bepakol egy eseményt a naptárba. Nincs sok ragozni való rajta, ez is láncolt listát manipulál: eldönti, hogy hova szúrjon be és oda teszi a paraméterként kapott eseményt (azazhogy csinál egy új listaelemet és azt teszi be, illetve ahhoz rendeli a kapott `VEvent`-et)

2.2.6 `MYERRNO Calendar_deleteVEvent(Calendar* cal, VEvent ve)`

Az előző párja. Ez egy eseményt töröl, de úgy, hogy egy eseményt kap, amit összehasonlít az összessel a naptárban, és ha egyezést talál, akkor azon a helyen töröl. Nem a legeffektívebb megoldás, de működik, illetve bizonyos helyzetekben nagyon jól jön, hogy így van megoldva.

2.2.7 `MYERRNO VEvent_delete(VEvent* ve)`

Nagyon rövidke függvény, ami felszabadítja egy `VEvent` típusú változóban a dinamikus karaktertömböket.

2.3 `interactiveHandler.c`

Ez a fájl az interaktív, szöveges mód függvényeit tartalmazza, azonban úgy alakult, hogy tulajdonképpen mindenféle I/O-ra ezek a függvények használódnak. Fontosabb függvények:

2.3.1 void shell(void)

Az egész `shell` dolog egy leírást követve készült, ami a forrásfájlban be van linkelve. Nem szeretném túl részletezni, lényegében egy parancssor feladatait látja el, mármint egy nagyon alapszintű parancssorét. Nincsenek pipeline-ok, vagy ilyesmik. De parancsokat meg lehet hívni, argumentumokkal és a program értelmezi őket.

2.3.2 void shell_say(ShellSays ss, const char* message, ...)

Talán ez a legtöbbet használt függvény, de nem csinál semmi igazán fontosat. Mindössze kapcsolatot tart fenn a felhasználóval. Egy üzenetet kap, amit formázottan ki tud tenni a standard kimenetre. Meg lehet neki adni, hogy milyen jellegű az üzenet (figyelmeztetés, hiba, kérdés, stb.) és aszerint tesz egy kis szimbólumot (Tehát itt cserélhetőek ezek és mindenhol úgy fog kiírni, ha átírjuk. Szóval itt is egy konfigurációs lehetőség)

Van még itt egy pár bemeneti függvény, ami szöveget, időpontot, számot olvas be... Ezeket nem részletezem, egészen érhetőek a kódból is. Itt van még a sok helyen meghívott hibakezelő függvény, ami `MYERRNO`-t kap és valami üzenetet csinál belőle.

2.4 commandHandler.c

Pokoli hosszú, több mint 1000 soros fájl. A parancsok függvényeit tartalmazza. Valószínűleg sokat lehetne dobni rajta, ha effektívebben lennének kezelve az argumentumok és nem a jelenlegi megoldással. Nem sorolom fel a függvényeket, mert mire leírnám mindegyiket, mármint működésüket, a 100. oldalt írnám. A lényeg annyi, hogy mindegyik kezeli a kapott argumentumait és egyéb függvényekkel manipulál. A help parancs kiadásával látható, milyen parancsokat, hogyan lehet használni, de ez benne van a felhasználói dokumentációban

2.5 GUIHandler.c

Az ominózus grafikus felület kezelője. Nagyon röviden megmagyarázható, hogyan működik, hiszen egyszerűen `GTK+3.0` könyvtári függvényeket használ. Néhol egy-egy kódrészlet el van csippentve korábbi függvényekből és kicsit rá van formálva a `GTK`-val való használatra. Vannak hibák, például az új esemény létrehozása ebben a módban még nem is működik.

2.6 helper.c

Egy másik őrült hosszú fájl. Itt nincsen külön említésre méltó függvény, mindegyik viszonylag rövid és egyértelmű feladatot lát el. A legtöbb helyen a kommentek nagyban segítik a tájékozódást.

3 Adatstruktúra, tudnivalók

Nem túl bonyolult a naptár kialakítása a memóriában. Az egész naptárat meghatározza egy `Calendar` típusú változó, ami egy kétszeresen láncolt, strázsás lista paramétereit tartalmazza: ezek a lista elemeinek száma (a strázsák nélkül), a lista első strázsája és a másik. Üresen létrehozott `Calendar`-ban a strázsák egymásra mutatnak. A fentebb leírt adatbázis kezelő függvények végzik innentől kezdve a feladatokat. Egyszerűen csak a láncol listához kell hozzáfűzni, kivenni...

Azért ilyen adatszerkezetet választottam, mert az első stádiumban még jónak tűnő dinamikus tömbös megoldás nem vezetett jó eredményre. Sok esemény esetén ide-oda kellett volna tologatni a tömb elemeit a különböző parancsok kiadására. Ez sok erőforrásba került volna és kevésbé is lett volna elegáns megoldás. A láncolt listával ez a probléma megszűnik és csak pointer-ekkel kell manipulálni.

Említésre méltó a `MYERRNO` enum. Igazából semmi jelentős, mindössze egy egyéni hibakezelési megoldás, ami hasonlít az `errno`-ra. Könnyen kezelhető, mert csak annyit kell csinálni, ha új fajta hiba jön elő például, hogy hozzá kell adni egy elemet az `enum`-hoz, majd meg kell írni a hibakezelő függvény (`shell_errorHandler()`) viselkedését a hibára. Ez igazából egy hibaüzenetet jelent.

Nem kell sokat magyarázni, hogyan lehet használni a kódot programozóként: szinte végig primitív, alacsonyszintű függvények végzik a munkát, szerintem egészen érhető, hogy hogyan. A bonyodalmat mindössze a nehézkesz szövegalapú UI (TUI) generálja, de az elkerülhetetlen, sajnos.

A felhasználói dokumentációban olvasható, hogy hogyan lehet használni a drunt-ot.