Írjunk saját grepet!

A GNU libc Posix-megfelelő, szabályos kifejezések kezelésére alkalmas függvényeinek bemutatása.

Előfordult már veled, hogy egy felhasználó által megadott kifejezésre történő keresést akartál megvalósítani C-programodban? Ha igen, és eddig még nem találtad meg a feladat szép megoldását, ez a cikk neked szól. A méltán népszerű `grep` nevű segédprogram által szintén alkalmazott szabályos mintaillesztés az egyik leghatékonyabb eszköz a haladóbb felhasználók kezében. Szabályos kifejezésekkel (regular expressions) pillanatok alatt hámozhatunk ki átláthatatlan méretű szöveges adathalmazból értékes adatokat.

Minden Linux-terjesztés része a GNU `libc6`, amely a Posix-szabványhoz erősen igazodik. A GNU `libc` a ***regex.h*** fejléccállománnyal szolgál a szabályos kifejezések adta korlátlan lehetőségek kiaknázásához. Az első és legfontosabb általunk is használt elem a `regex_t` típus. Ez egy szerkezettípus, mely a szabályos kifejezésünket fogja tárolni, illetve számos, belső használatra fenntartott tárolót fog össze. Közvetlenül tilos belenyúlni, csupán a megfelelő függvényhívásokon keresztül szabad hozzáférni a tartalmához. Ennek megfelelően, miután létrehoztunk egy `regex_t` típusú változót, a szabályos kifejezést bele kell töltenünk. Ezt valósítja meg a `regcomp()` függvény. Három jellemzője közül az első egy mutató egy `regex_t` típusra. Ez nyilvánvalóan a fent említett változónk címe lesz. A második maga a szabályos kifejezés egy karakterlánc formájában. Az utolsó pedig egy egész szám, mellyel a mintaillesztés módját befolyásolhatjuk.

A ***regex.h*** számos makróval áll rendelkezésünkre. Ezekkel bitenkénti vagy művelettel összekapcsoltan több beállítást hajthatunk végre egyszerre. Két igen fontosat emelnék ki a sorból: az egyik a `REG_EXTENDED`, amely továbbfejlesztett mintaillesztést ír elő. Ez hasonló működést jelent, mint amit az `egrep`-pel tapasztalhatunk. A másik a `REG_ICASE`, amellyel a kis- és nagybetűk közötti különbséget hagyjuk figyelmen kívül.

A feltöltött szerkezetet dolgunk végeztével fel kell szabadítanunk. Az ezt megvalósító `regfree()` egyetlen értéket vár. Egy mutatót kér a `regex_t` típusra, tehát itt is a szabályos kifejezésünket tároló szerkezet címét fogjuk átadni.

A tényleges munkát végző függvény neve `regexec()`. Ijesztően hangzó öt kapcsolója közül az utolsó három többnyire alapértelmezés szerinti értéket kaphat. Először szokás szerint szerkezetünk címét adjuk át. Másodikként azt a karakterláncot, amire el akarjuk végeztetni a mintaillesztést. A harmadik és negyedik kapcsoló a részekre bontáshoz szükséges. Amennyiben nem kívánunk élni ezzel a lehetőséggel, a `0`, illetve `NULL` értékek lehetnek. Utolsóként további makrók segítségével egyéb kapcsolókat állíthatunk be az illesztéshez.

Sokszor háttérbe szoruló, ám annál fontosabb a család negyedik és egyben utolsó tagja, a `regerror()`. Ez a `regcomp()` vagy a `regexec()` visszatérési értékéhez szolgáltat hibaüzenetet. Első értéke a hibakód. A második annak a szerkezetnek a mutatója, amellyel a hiba előfordult. A harmadik a hibaüzenet tárolására szánt átmeneti tár, míg a negyedik ennek a mérete.

Hogyan lesz mindebből program? Példaképpen készítsük el saját `egrep`-ünket! A feladatot annyiban egyszerűsítjük, hogy nem értelmezünk kapcsolókat, illetve az adatot kizárólag a szabványos bemenetről fogadjuk. A továbbfejlesztett mintaillesztés szabályait fogjuk követni, ahogy az `egrep` is teszi. Egyetlen korlátozással fogunk élni, miszerint feltesszük, hogy egy sor nem hosszabb 255 karakternél.

```
#include <stdio.h>
#include <regex.h>
#include <sys/types.h>

#define MAXHOSSZ 256

int main (int argc, char **argv)
{
    char sor[MAXHOSSZ];
    int hiba;
    regex_t regkif;

    if (2 > argc) {
       fprintf (stderr, "Hasznalat: %s
                ➥{regkif}\n", argv[0]);
       return 1;
    }

    hiba = regcomp (&regkif, argv[1],
           ➥REG_EXTENDED);

    if (hiba) {
```

```
      regerror (hiba, &regkif, sor, MAXHOSSZ);
      fprintf (stderr, "Hiba: %s\n", sor);
      return 2;
    }

    while (NULL != fgets (sor, MAXHOSSZ, stdin)) {
      if (0 == regexec (&regkif, sor, 0, NULL, 0))
          printf ("%s", sor);
    }

    regfree (&regkif);

    return 0;
}
```

Sorról-sorra tekintsük át a program működését! A # (kettős kereszttel) kezdődő sorokat az előfeldolgozó fogja értelmezni. Három fejlécállományt használunk, az ***stdio.h***-t a be- és kimenetet kezelő függvényekhez, a ***regex.h***-t a szabályos mintaillesztés függvényeihez, illetve a ***sys/types.h***-t. Ezen túlmenően meghatározunk egy makrót `MAXHOSSZ` néven, 256 értékkel. Ez adja meg annak az átmeneti tárnak a méretét, melyben a sorokat tárolni fogjuk.
A `main()` fejlécében az `argc` változó az `argv` elemszámát tartalmazza. Az `argv` nulladik eleme az a név, amellyel meghívták a programot. További elemei a program esetleges kapcsolói. A `sor` változónk a fő ciklusunk minden lefutása során egy sort fog tartalmazni a bejövő adatokból. Fontos, hogy legfeljebb `MAXHOSSZ` – egy karakter tárolására alkalmas, mivel a karakterláncot lezáró 0-t is tárolni kell.
A hiba a `regcomp()` visszatérési értékét fogja megkapni. Végül pedig a `regkif`, a szabályos kifejezésünk szerkezete. Az első `if` feltételben az átadott értékek számát ellenőrizzük. Az `argc` mindig legalább 1, mivel az `argv` nulladik eleme, a program neve mindig adott. Ezért az `argc` 2 lesz, ha a program egy értéket kapott, így a feltételünk akkor igaz, ha értékek nélkül indítottuk a programot. Ebben az esetben a szabványos hibacsatornára írjuk ki a program használatának helyes módját. Ezt követően 1 visszatérési értékkel véget ér a program.
Ha kaptunk értéket, a `regcomp()`-ot futtatjuk vele. A `regkif` címét az & (és) karakterrel képezzük, ez a függvény első értéke. Másodikként a programnak átadott szabályos kifejezést kapja. Harmadikként megadjuk a `REG_EXTENDED` makrót, mellyel továbbfejlesztett mintaillesztésre utasítjuk a függvényt. A `regcomp()` visszatérési értékét a `hiba` változóban tároljuk. A `hiba` nullától különböző értéket fog kapni, ha a szerkezetbetöltés sikertelen volt.
Amennyiben hiba történt, meghívjuk a `regerror()` függvényt. Első értékként a hibakódot kapja, majd a szerkezet címét. A harmadik érték az az átmeneti tár, amelyben a hibaüzenetet kívánjuk tárolni. E helyen felhasználjuk `sor` változónkat. Ennek hosszát ismerjük, a `MAXHOSSZ` makróval állítottuk be, ez lesz a negyedik érték. Miután a `sor` felvette a hibaüzenetet, a szabványos hibacsatornára írjuk ki az üzenetet, majd 2 visszatérési értékkel kiszállunk.
Ha minden rendben ment, jöhet a fő ciklus. Mindenekelőtt el kell döntenünk, hogy milyen függvényt használunk a szabványos bemenet soronkénti olvasására. Első kóbor ötletünk a `gets()` lehetne, de használata nem ajánlott. Semmiféle ellenőrzést nem tartalmaz az átmeneti tár túlírásra, így könnyen elszállhat a programunk, amellett, hogy nem is biztonságos. Első ránézésre kényelmetlennek tűnhet az `fgets()` alkalmazása, mert egy felső korlátot kell megadni a beolvasandó adatok méretére, de pont ez adja az előnyét is. Az `fgets()`-szel a `sor` tömbbe egy új sort olvasunk be a sortörés jellel együtt, a végén a lezáró nullával. A visszatérési érték a sor címe vagy `NULL`, ha a bejövő adat elfogyott. Ezzel kiváló ciklusfeltételt adtunk meg. A `regexec()`-et csak alapszinten használjuk, a szerkezet címét és a soron következő sort adjuk meg neki. Ha a visszatérési érték 0, akkor illeszkedik. Ebben az esetben kiíratjuk `sor` változót. Itt is megfigyelhetjük, hogy az `fgets()` a sortöréssel együtt olvassa be az adatot.
A program végén felszabadítjuk a szerkezetnek azt az elemeit, amelyeket a `regcomp()` dinamikusan foglalt, majd 0 visszatérési értékkel kilépünk.
Lássuk hogyan használható frissen írt programunk! A C-forrást előbb le kell fordítanunk. Mindig érdemes kitenni a `-Wall` (warning all) kapcsolót, mellyel arra kérjük a fordítót, hogy az összes figyelmeztetést jelenítse meg.
Alapértelmezésben ugyanis kizárólag a hibákat (errors) írja ki a képernyőre. Ennek megfelelően:

```
$ gcc -Wall -o regkif regkif.c
```

Így létrejött egy ***regkif*** nevű állomány, amit a fordító okosan már futtathatóvá is tett. Először nézzük meg, működnek-e a kivételkezelő részek. Először indítsuk el kapcsolók nélkül a programot:

```
$ ./regkif
Hasznalat: ./regkif {regkif}
```

Lássuk, hogyan működik a `regerror()`! Figyeljük a választ a különböző hibás szabályos kifejezések megadása esetén:

```
$ ./regkif "*"
Hiba: Invalid preceding regular expression
$ ./regkif "\1"
Hiba: Invalid back reference
$ ./regkif "[[:alma:]]"
Hiba: Invalid character class name
```

Most már jöhet egy éles próba. A */etc/services* fájlt fogjuk használni. Ez egy egyszerű szöveges fájl, ami hálózati szolgáltatásokhoz rendel kapuszámokat. Innen tudhatjuk meg, hogy az FTP szokásos kapuja a 21, a HTTP-é a 80. Remekül lehet benne keresni. Tegyük fel, hogy mi arra vagyunk kíváncsiak, hogy az 53-as mi szokott lenni.

```
$ cat /etc/services | ./regkif
"^[[:alnum:]]+[[:blank:]]+53/tcp"
domain  53/tcp  nameserver   #name-domain server
```

Számos lehetőséget nem ismertettem a helyszűke okán, ezért mindenképpen érdemes elolvasni a regex(3) útmutatásait.
Ami a legfontosabb: soha ne találd fel újra a kereket!

Fülöp Balázs