REGULARNI IZRAZI U C-u

Kao što je to slučaj pri radu u Unix shellu, odnosno sa shell skriptama, regularni izrazi pokazuju se potrebnim i korisnim i u C programima. Za rad s regularnim izrazima u C-u postoji biblioteka regex koju treba includati u program da bi nam postale dostupne odgovarajuće funkcije. Opis ovih funkcija može se naći na man stranici biblioteke regex (`man regex`). Ovdje ćemo opisati te funkcije malo prodrobnije i dati nekoliko primjera njihovog korištenja. Inače, još neki detalji specifikacije ovih funkcija, koje ovdje nećemo navoditi, mogu se naći na stranici **http://pubs.opengroup.org/onlinepubs/007908799/xsh/regcomp.html**.
Funkcije koje sadrži biblioteka regex su sljedeće:

```
int regcomp(regex_t *preg, const char *pattern, int cflags);

int regexec(const regex_t *preg, const char *string, size_t nmatch,
    regmatch_t pmatch[], int eflags);

size_t regerror(int errcode, const regex_t *preg, char *errbuf,
    size_t errbuf_size);

void regfree(regex_t *preg);
```

Funkcija `regcomp` služi za "kompajliranje" regularnog izraza tj. za pretvaranje odgovarajuće string varijable (`pattern`) u strukturnu varijablu za `regex_t` (`preg`), koja se onda koristi za usporedbu sa zadanim stringom (`string`), što se vrši pomoću funkcije `regexec`. Dakle, ako u svom C programu želimo vidjeti da li regularni izraz "zahvaća" neki string ili ne, trebamo najprije pomoću funkcije `regcomp` "kompajlirati" taj regularni izraz, zatim objekt (strukturnu varijablu) koji smo dobili na taj način staviti kao argument funkcije `regexec` skupa sa spomenutim stringom, pa na kraju, nakon njezinog "procesiranja", vidjeti što je ova funkcija vratila - ako je vratila 0, to znači da je odgovor na gornje pitanje pozitivan (pattern matching). Napomenimo da funkcija `regexec` također vraća 0 ako je kompajliranje regularnog izraza prošlo uspješno.
Evo primjera C-programa koji radi jednu takvu usporedbu. Program se kompajlira (na Debian Linuxu) po standardnom postupku, bez uvrštavanja nekih dodatnih (statičkih/dinamičkih) biblioteka, a pokreće se uz dva argumenta - prvi je regularni izraz, dok je drugi string koji se s njim uspodređuje (u kodu smo radi kratkoće izostavili provjeru jesu li argumenti zadani ili ne).

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

int main(int argc, char *argv[])
{
   regex_t regex;
   int reti;
   char msgbuf[100];

/* Compile regular expression */
   reti = regcomp(&regex, argv[1], REG_EXTENDED | REG_ICASE);
   if( reti )  {
      fprintf(stderr, "Could not compile regex\n");
      exit(1);
   }

/* Execute regular expression */
   reti = regexec(&regex, argv[2], 0, NULL, 0);
   if( !reti )
      puts("Match");
   else if( reti == REG_NOMATCH )
      puts("No match");
```

```
   else  {
      regerror(reti, &regex, msgbuf, sizeof(msgbuf));
      fprintf(stderr, "Regex match failed: %s\n", msgbuf);
      exit(1);
   }

/* Free compiled regular expression if you want to use the regex_t again */
   regfree(&regex);

   return 0;
}
```

Program izbacuje poruku o tome da li regularni izraz iz prvog argumenta zahvaća string iz drugog ili ne. Treba reći da osim spomenute provjere argumenata, ovdje prikladno nije riješena niti stvar sa porukom o mogućoj grešci koja se sprema u varijablu `msgbuff` ("message buffer", vidi argumente funkcije `regerror`). Uočimo da je za tu varijablu unaprijed alocirano 100 karaktera (byteova), premda njezina duljina može varirati. No, treba napomenuti da funkcija `regerror`, koja omogućava da se na osnovi error kodova što ih vraćaju funkcije `regcomp` ili `regexec` poruka o greški spremi u message buffer (argument `errbuf` u njezinoj definiciji), sama vraća upravo duljinu poruke o greški (zajedno sa terminirajućom nulom), pa ovaj problem možemo riješiti dvostrukim pozivom funkcije `regerror` - najprije ju pozivamo za "prazan" message buffer (posljednji argument ove funkcije (`errbuf_size`) predstavlja, jasno, veličinu ovoga stringa), čime utvrđujemo njegovu duljinu, a zatim prikladno alociramo message buffer i "punimo" ga ponovnim pozivom funkcije `regerror`. Dakle, zadnji poziv funkcije `regerror` u gornjem kodu zgodno je supstituirati sa slijedeće dvije linije:

```
msgbuf = malloc(regerror(reti, &regex, (char *)NULL, (size_t)0));
regerror(reti, &regex, msgbuf, sizeof(msgbuf));
```

Funkcija `regfree` oslobađa prostor u memoriji koji je zauzela (strukturna) varijabla `regex`, definirana putem kompajla zadanog regularnog izraza.
Recimo još nešto o ostalim parametrima naredbi `regcomp` i `regexec`, dakle onima koji dosad nisu bili spomenuti. U slučaju funkcije `regcomp` integerski parametar `cflag` može imati slijedeće vrijednosti:
`REG_EXTENDED` - Koristi se sintaksa proširenih (extended) POSIX regularnih izraza (ERE) pri interpretaciji varijable `regex`. Ako nije definirano, koristi se osnovna (basic) sintaksa (BRE). Inače, razlika osnovnih i proširenih regularnih izraza je u suprotnom značenju escapinga nekih metakaraktera (zagrada) i u tome što prošireni regularni izrazi obuhvaćaju i neke dodatne metakaraktere (?, +, |) - detalje o ovoj razlici vidi u odgovarajućoj dokumentaciji.
`REG_ICASE` - Ne razlikuju se velika i mala slova pri usporedbi. Ovdje treba biti oprezan sa non-ASCII karakterima (recimo "našim slovima"), gdje ovo obično ne funkcionira.
`REG_NOSUB` - Izostavljena podrška za pronalaženje substringova. U slučaju kad je varijabla `regex` kompajlirana uz ovaj parametar, argumenti `nmatch` i `pmatch` pri pozivu funkcije `regexec` za istu tu varijablu se ignoriraju.
`REG_NEWLINE` - Metakarakteri koji predstavljaju bilo koji karakter ne zahvaćaju znak newline (ASCII 10). Također, non-matching lista (`[^...]`) koja ne sadrži znak newline, ne zahvaća taj znak.
U argumentu funkcije `regcomp` može se navesti jedan ili više ovih parametara povezanih operatorom bitwise OR (|).
Što se tiče funkcije `regexec` integerski parametar `cflag` može imati slijedeće vrijednosti:
`REG_NOTBOL` - Metakarakter koji označava početak linije (^) ne zahvaća početak stringa. Ovaj parametar se koristi kad se uspoređuju različiti dijelovi stringa, pa početak stringa ne predstavlja i početak linije.
`REG_NOTEOL` - Metakarakter koji označava kraj linije ($) ne zahvaća kraj stringa. Ovaj parametar se koristi kad se uspoređuju različiti dijelovi stringa, pa kraj stringa ne predstavlja i kraj linije.
Ovdje treba napomenuti da u slučaju kad je varijabla `regex` kompajlirana sa parametrom **`REG_NEWLINE`**, metakarakter (^) zahvaća prazan string poslije znaka newnine, a metakarakter ($) zahvaća prazan string prije znaka newnine.
Što se tiče argumenata `nmatch` i `pmatch`, oni se odnose na zahvaćanje "podizraza" (subexpressions, koji se u regularnim izrazima označavaju okruglim zagradama) i u slučaju kad nas podizrazi ne zanimaju za njihovu vrijednost treba staviti 0 odnosno NULL. Ove vrijednosti spomenuti argumenti automatski dobivaju (kao što

je već spomenuto) u slučaju kad je varijabla `regex` kompajlirana sa parametrom **`REG_NOSUB`** ("deaktiviranje" rada s podizrazima). Inače, `pmatch` predstavlja polje (array) strukturnih varijabli za `regmatch_t`, što je struktura koja sadrži pozicije (offsete) početka i kraja podizraza regularnom izrazu koji su pronađeni u zadanom stringu. Ova struktura definirana je (u `regex.h`) na slijedeći način:

```
typedef struct
{
   regoff_t rm_so;
   regoff_t rm_eo;
} regmatch_t;
```

Pozicije (offseti) početaka pronađenih podizraza (substringova u zadanom stringu) spremaju se u (integerski) strukturni element `rm_so`, a pozicije njihovih završetaka (krajeva) u `rm_eo` - točnije u ovaj drugi element se sprema pozicija karaktera koji slijedi nakon kraja substringa. Pritom `pmatch[0]` odnosi na identifikaciju substringa koji odgovara cijelom regularnom izrazu, a brojanje pozicija počinje od nule (prvom karakteru odgovara nulta pozicija). Vrijednosti ovih varijabli pronalaze se odnosno definiraju tokom "procesiranja" koje izvodi funkcija `regexec`.
Ako je broj podizraza u regularnom izrazu manji od `nmatch`, onda se za vrijednosti strukturnih elemenata `rm_so` i `rm_eo` u ostalim (suvišnim) strukturnim varijablama polja stavlja -1.
Ovu priču možemo pojasniti na sljedeća dva primjera. U prvom primjeru polje strukturnih varijabli ima samo jedan element (`nmatch = 1`), a regularni izraz ne mora sadržavati podizraze (njihovo definiranje nema ni smisla). Primjer također demonstrira i upotrebu parametra **`REG_NOTBOL`** u funkciji `regexec`. Ovaj C-program pokrećemo, isto kao i onaj prethodni, uz dva argumenta - regularni izraz i string koji se s njim uspoređuje (u ovom je primjeru također, radi kratkoće izostavljena provjera argumenata programa).

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <regex.h>
#include "psutl.c"

int main(int argc, char *argv[])
{
   regex_t regex;
   regmatch_t pm;
   int i = 0;
   int reti;
   int nwst = 0;

/* Execute regular expression */
   regcomp (&regex, argv[1], 0);

/* This call to regexec() finds the first match in the line. */
   reti = regexec (&regex, argv[2], 1, &pm, 0);
   while (reti == 0) {  /* While matches found. */
      i++;
      printf("Match %d. pos = %d\n", i, nwst + pm.rm_so);
      nwst +=  pm.rm_eo;
/* This call to regexec() finds the other matches in the line. */
      reti = regexec (&regex, argv[2] + nwst, 1, &pm, REG_NOTBOL);
   }

/* Free compiled regular expression if you want to use the regex_t again */
   regfree(&regex);

   return 0;
}
```

Kao što vidimo, ovaj će program izbaciti sve pozicije uzorka definiranog regularnim izrazom iz prvog argumenta, u stringu koji je zadan kao drugi argument.

U drugom primjeru polje strukturnih varijabli ima 4 elementa (`nmatch = 4`), i tu se pronalaze pozicije još triju substringova što odgovaraju podizrazima u regularnom izrazu zadanom kao prvi argument programa (drugi argument je opet string za usporedbu, a priča s izostavljenom kontrolom argumenata programa također se ponavlja).

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

int main(int argc, char *argv[])
{
   regex_t regex;
   regmatch_t pm[4];
   int reti;

/* Execute regular expression */
   regcomp (&regex, argv[1], REG_EXTENDED);

/* This call to regexec() finds three matching substring offsets in the line. */
   reti = regexec (&regex, argv[2], 4, pm, 0);
   printf("Match = %d, Substrn_Positions = %d, %d, %d, %d\n", reti, pm[0].rm_so,
 pm[1].rm_so, pm[2].rm_so, pm[3].rm_so);
/* Free compiled regular expression if you want to use the regex_t again */
   regfree(&regex);

   return 0;
}
```

Napomenimo na kraju da se neka dodatna pravila za zahvaćanje substringova čiji se offseti spremaju u polje `pmatch[]` mogu naći na gore navedenoj web stranici. Tu se također može naći popis grešaka pri kompilaciji regularnog izraza (error return values za `regcomp`) i još neke druge informacije. Svakako je korisno pročitati i man stranicu za biblioteku regex (`man regex`), a na internetu se može pronaći i dosta druge korisne dokumentaciju vezane uz ovu temu.
O regularnim izrazima općenito vidi **http://pubs.opengroup.org/onlinepubs/007908799/xbd/re.html**. Ova stranica, uz onu već spomenutu, predstavlja i glavnu referencu ovoga teksta, s tim da se tu može dodati i članak o regularnim izrazima iz Wikipedije.