Professur
Betriebssysteme

Algorithmen und Programmierung

Reguläre Ausdrücke (Trainingsaufgabe)

Stellen Sie sich vor, Sie bekommen eine wichtige Email mit der Kontaktrufnummer, wo aus irgendeinem technischen Fehler alle Symbole (bis auf die Symbole von den o.g. Telefonnumer) gemischt sind. Der Text ist zu lang, um den selbst zu analysieren und die wichtige Rufnummer zu finden. Allerdings wissen Sie, dass die Rufnummer wie folgt aussehen soll:

1. Eine 0 oder das Landcode, bestehend aus einem +-Zeichen und einer bis zwei Ziffer;
2. Dem Landcode folgt immer ein Trennzeichen: entweder ein Leerzeichen oder ein Minuszeichen;
3. Stadtcode, der aus drei bis fünf Ziffern besteht;
4. Ein Trennzeichen: entweder ein Leerzeichen oder ein Minuszeichen;
5. Die eigentliche Nummer mit mindestens sechs und höchstens acht Ziffer;
6. Die erste Ziffer von den Landcode, Stadtcode und der Nummer darf nicht 0 sein.

Somit haben die folgenden Rufnummern die richtige Struktur:

1. 0154␣2365890
2. 0154-2365890
3. 012345-12345678
4. +49-154␣2365890
5. +49␣154-2365890
6. +10␣12345␣12345678

Ungültige Rufnummern sind z.B.:

1. 0154␣␣2365890
2. 0154␣236␣589␣0
3. 0154-02365890
4. 0-154–2365890
5. 0123456-12345678
6. +49154␣2365890

7. +491542365890
8. +01-02345-02345678

In solchen Situationen bietet es sich an, die Textelemente mithilfe von regulären Ausdrücken zu finden. In C gibt es mehrere Bibliotheken, die reguläre Ausdrücke unterstützen. Ihnen stehen die unten beschriebene Funktionen zur Verfügung. Machen Sie sich mit den Funktionen der Bibliothek regex.h vertraut. Ihre Aufgabe ist, die Zeile mit dem regulären Ausdruck (Zeile 79 im Code) so zu ergänzen, damit die Rufnummer mit der o.g. Struktur gefunden werden kann.

Beispiel. Bei der Eingabe "asddasdadada0176-23880123adadscata. qwenm234234asdsajpkmqnjqheoijwm +49-176-2301320000 adasdada" soll die Ausgabe wie folgt sein:

```
$& is '0176-23880123' (bytes 12:25)
$& is '+49-176-23013200' (bytes 68:84)
```

Bei der zweiten Rufnummer versucht der Pattern Matcher immer die maximale Anzahl der Symbole zu finden, deswegen besteht das letzte Teil der Rufnummer aus acht, und nicht sechs Ziffer. Bitte beachten Sie, dass die Ausgabe schon formatiert ist, Sie müssen also nur die o.g. Struktur in Form eines gültigen regulären Ausdruckes beschreiben[1].

```c
// Code original available on: https://www.lemoda.net/c/unix-regex/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <regex.h>

/* The following is the size of a buffer to contain any error messages
   encountered when the regular expression is compiled. */

#define MAX_ERROR_MSG 0x1000

/* Compile the regular expression described by "regex_text" into
   "r". */

static int compile_regex (regex_t * r, const char * regex_text)
{
    int status = regcomp (r, regex_text, REG_EXTENDED|REG_NEWLINE);
    if (status != 0) {
  char error_message[MAX_ERROR_MSG];
  regerror (status, r, error_message, MAX_ERROR_MSG);
        printf ("Regex error compiling '%s': %s\n",
                regex_text, error_message);
        return 1;
    }
    return 0;
}

/*
  Match the string in "to_match" against the compiled regular
  expression in "r".
 */
```

[1]Mehr Informationen zu der verwendeten Bibliothek finden Sie unter `http://www.wellho.net/mouth/2844_Learning-about-Regular-Expressions-in-C-through-examples.html` und `https://en.wikibooks.org/wiki/Regular_Expressions/POSIX_Basic_Regular_Expressions`

```c

static int match_regex (regex_t * r, const char * to_match)
{
    /* "P" is a pointer into the string which points to the end of the
       previous match. */
    const char * p = to_match;
    /* "N_matches" is the maximum number of matches allowed. */
    const int n_matches = 1;
    /* "M" contains the matches found. */
    regmatch_t m[n_matches];

    while (1) {
        int i = 0;
        int nomatch = regexec (r, p, n_matches, m, 0);
        if (nomatch) {
            printf ("No more matches.\n");
            return nomatch;
        }
        for (i = 0; i < n_matches; i++) {
            int start;
            int finish;
            if (m[i].rm_so == -1) {
                break;
            }
            start = m[i].rm_so + (p - to_match);
            finish = m[i].rm_eo + (p - to_match);
            if (i == 0) {
                printf ("$& is ");
            }
            else {
                printf ("$%d is ", i);
            }
            printf ("'%.*s' (bytes %d:%d)\n", (finish - start),
                    to_match + start, start, finish);
        }
        p += m[0].rm_eo;
    }
    return 0;
}

int main (int argc, char ** argv)
{
    regex_t r;
    const char * regex_text;
    const char * find_text;

    regex_text = ""; // hier anpassen
    if (argc < 2) {
        printf("Not enough data!");
    }
    else {
        find_text = argv[1];
    }
    printf ("Trying to find '%s' in '%s'\n", regex_text, find_text);
    compile_regex (& r, regex_text);
    match_regex (& r, find_text);
    regfree (& r);
```

```
90      return 0;
91  }
```

Listing 1: regex.c

Reichen Sie ihre angepasste Datei `regex.c` als Lösung ein.

Hinweise zur Aufgabenstellung

Für die Lösung dieser Aufgabe benötigen Sie folgende Grundkenntnisse. Im in Klammern angegebenen Foliensatz finden Sie Informationen dazu.

- Schreiben von Funktionen (Kapitel III)
- Kontrollfluss in C (if) (Kapitel III)
- Schleifen in C (Kapitel V)
- Felder in C (Kapitel IV)
- Datenstrukturen (struct) in C (Kapitel IV)
- Benutzung von `gcc` (Kapitel III)

Hinweise zur Abgabe

- Erstellen Sie eine ZIP- bzw. TGZ-Archivdatei, welche die geforderten Dateien enthält.
- Fügen Sie dem Archiv keine weiteren Dateien oder Ordner hinzu.
- Reichen Sie Ihre Lösung unter `https://osg.informatik.tu-chemnitz.de/submit` ein.
- Bis zum Abgabeende (Deadline), sofern gegeben, können beliebig neue Lösungen eingereicht werden, die die jeweils älteren Versionen ersetzen.
- Ihr Programm muss auf der Testmaschine übersetzbar sein. Deren Details sind auf dem OpenSubmit-Dashboard verfügbar.
- Ihre Lösung wird automatisch validiert. Sie werden über den Abschluss der Validierung per eMail informiert.