strstr() + CLI

Montag, 14. März 2022 10:37

1) Eine 'elegante' Möglichkeit für ein Softwaretest von einzelnen Funktion ist die Nutzung eines Kommandozeileninterpreters (Command-Line Interpreter CLI) (siehe auch https://de.wikipedia.org/wiki/Kommandozeileninterpreter). Ein Kommandozeileninterpreter ist ein Programm, das eine Benutzereingaben zeilenweise von einer Kommandozeile einliest, diese Eingabe in Tokens aufteilt und in Abhängigkeit des ersten Tokens eine entsprechende Funktionalität ausführt. Die nachfolgenden Tokens dienen dann der Funktionalität als Eingabeparameter.

   Folgende Funktionalitäten soll der zu erstellende Kommandozeileninterpreter darstellen:

```
help                    --> Ausgabe dieser Hilfsmeldung
quit                    --> Ende des Programms
printstr par1           --> Aufruf der Funktion printstr()
quersumme par1          --> Aufruf der Funktion quersumme()
strstr haystack needle --> Aufruf der selbst geschriebenen Funktion
strstr
```

   Das Rahmenprogramm zur Lösung dieser Aufgabe ist vorgegeben. Innerhalb einer while(1) Schleife wird mittels fgets() eine Eingabe vom Benutzer angefordert und diese im String input zurückgegeben. Im Anschluss wird von der Eingabe mittels strtok() das erste Token abgetrennt.
   Vervollständigen sie das Rahmenprogramm, d.h. separieren sie die nachfolgenden Tokens (Eingabeparameter), und rufen sie die geforderten Funktion unter Übergabe der passenden Eingabeparameter auf.

   Hinweis:
   - Neben der eigentlichen Implementierung geht es hier auch darum, die Rückgabewerte von Funktionsaufrufen zu nutzen und im Fehlerfall entsprechend zu reagieren. In C gibt es kein TryCatch, so dass sie dies Händisch übernehmen müssen!
   - Das drücken der ENTER-Taste am Ende jeder Zeile bedingt, dass auch dieses Zeichen (Zeilenvorschub '\n') in der Eingabe gespeichert wird. Bspw. wird bei der Eingabe von "quersumme 11" folgende Zeichen gespeichert:
     input[]={'q','u','e','r','s','u','m','m','e',' ','1','1','\n','\0'};
     Finden sie einen eleganteren als den hier realisierten Weg, dieses Zeichen zu 'eliminieren'.
   - Bei Nutzung des CompilerExplorers kann ein Eingabefenster über den Button 'Execution stdin' geöffnet werden. Hier können im Vorfeld mehrere Befehle eingetragen werden, die dann Zeilenweise abgearbeitet werden. Bswp.:

```
help
quersumme  1234
quersumme      47a11
strstr hallo  lo
strstr laaaaaaaangeZeile  >20Zeichen

quit
```

1) Schreiben sie eine eigene Implementierung von strstr()

```
char *strstr_own(char *haystack, char *needle) {

}
```

   Die Spezifikation besagt im Wesentlichen, dass diese Funktion nach dem ersten Auftreten des Strings needle im String haystack sucht. Der Vergleich findet ohne Berücksichtigung des

Stringendezeichens statt, so das needle auch innerhalb von haystack gefunden wird. Wenn needle gefunden wurde, gibt strstr die Startadresse des Substrings zurück (beispielsweise mit &haystack[4]), andernfalls die Konstante NULL.
Auszug aus der Original Spezifikation:
The **strstr**() function finds the first occurrence of the substring needle in the string haystack. The terminating null bytes (aq\0aq) are not compared.
Return value:
These functions return a pointer to the beginning of the substring, or NULL if the substring is not found.

Zur Selbstverdeutlichung der Arbeitsweise empfiehlt sich zunächst zu überlegen, was folgende Aufrufe zurückgeben:

```
char src[]="hallollo";
char *ptr1=strstr(src,"ll");
char *ptr2=strstr(src,"xy");
char *ptr3=strstr(src,"hallo123");
char *ptr4=strstr(src,"l");
char  *ptr5=strstr(src+4,"ll");
char *ptr6=strstr("ll",src);
```

Zur Vermeidung der Übergabe eines NULL Zeigers an die printf() Routine (dessen Verhalten dann undefiniert ist) empfiehlt sich für die Debug-Ausgabe folgende Anweisung:

```
printf("'%s'\n",ptr1?:"(Null-Pointer)");
```

Ergänzend haben sie die Möglichkeit, mittels des Kommandozeileninterpreters diese Funktion ausgiebig zu testen.

Hinweis:

- Neben der eigentlichen Implementierung geht es bei dieser Aufgabe auch darum ein Bewusstsein für Bufferoverflow zu schaffen. Bei unachtsamer Realisierung ist ein Bufferoverflow quasi vorprogrammiert (und damit ein Einfallstor für Viren vorhanden)!

```c
#include <stdio.h>    //fuer printf()  fgets()
#include <string.h>   //fuer strtok()
static int printstr(char *str,size_t n);
int quersumme(char *)
{
    int quer=0;
    //Ihre Realisierung aus der 1. Aufgabe
    return quer;
}
char *strstr_own(char *haystack, char *needle)
{
    printf("Suche in '%s' nach '%s'\n",haystack?:"(null)",needle?:"(null)");
    //Ihre Realisierung aus dem 2. Teil dieser Aufgabe
    return NULL;
}
static const char *help=
"help                   --> Ausgabe dieser Hilfsmeldung\n"
"quit                   --> Ende des Programms\n"
"printstr par1          --> Aufruf der Funktion printstr\n"
"quersumme par1         --> Berechnung der Quersumme aus dem 1. Parameter\n"
"strstr haystack needle --> Ohne worte\n";
int main(int argc, char *argv[])
{
    enum status {OK,KO_FGETS,KO_PRINTSTR,KO_END} ret=OK;
    printf("Programm: "
        "\e[31m"  //Farbe Rot
```

```
        __FILE__
        "\e[39m"  //Farbe Schwarz
        " \033[4m" //Unterstreichen ein
        "started\
        \033[0m\n"); //Unterstreichen aus
    while(1) {
        char input[20];
        //Zur Befehlseingabe im 'Executor' mittels des Buttons 'Execution
stdin'
        //den Eingabereich 'Execution stdin...' öffnen und dort ihre
        //Befehle eintragen. Die Eingabe kann mehrere Zeilen lang sein
        //fgets() liest diese zeilenweise ein!
        //In der letzten Zeile am besten quit eintragen!
        if(fgets(input,sizeof input,stdin)==NULL) {
            ret=KO_FGETS;
            break;
        }

        //Für sie als Hilfsfunktion zur Darstellung der fgets Rückgabe
        //(kann später gerne gelöscht werden)
        if(printstr(input,20)<0) {
            ret=KO_PRINTSTR;
            break;
        }
        char *arg1=strtok(input," ");

        //Für sie als Hilfsfunktion zur Darstellung der Arbeitsweise von
strtok()
        //(kann später gerne gelöscht werden)
        if(printstr(input,20)<0) {
            ret=KO_PRINTSTR;
            break;
        }
        if(arg1==NULL)
            continue;
        else if(strcmp(arg1,"quit\n"))
            break;
    }
    const char *retstring[]={
        [KO_FGETS]   ="Programm mangels Eingabe beendet",
        [OK]         ="Programm ordnungsgemäß beendet",
        [KO_END]     ="Unbekannter Rückgabewert",
        [KO_PRINTSTR]="printstr failed"
    };
    printf("%s\n",retstring[ret>KO_END?KO_END:(ret<0?KO_END:ret)]);
    return (int)ret;
}

static int printstr(char *str,size_t n)
{
    if(n==0)
        return printf("n muss größer als 0 sein!"),-1;

    for(size_t lauf=0;lauf<n;lauf++)
        printf("'%1c'%s",str[lauf]>=32?str[lauf]:'?',lauf==n-1?"\n":" ");
    unsigned char *ptr=str;
    for(size_t lauf=0;lauf<n;lauf++)
        printf("%02x %s",ptr[lauf],lauf==n-1?"\n":" ");
    return 0;
}
```