SNP: C Implementation eines Bash Scripts

```
path="$1"
[ -n "$path" ] || path="$PATH"

# input-field-separator: tells the shell to split in the 'for' loop the $var by ":"
IFS=":"

for p in $path
do
    i=$((i+1))
    [ -n "$p" ] || p="."
    if [ -d "$p" ] && [ -x "$p" ]
    then
        find -L "$p" -maxdepth 1 -type f -executable -printf "$i:%h:%f\n" 2>/dev/null
    fi
done
```

```
int main(int argc, const char *argv[])
{
    char *paths = malloc_copy(argc > 1 && argv[1][0] ? argv[1] : getenv("PATH"));
    // replaces all ':' by '\0' and returns the number of the resulting fields
    size_t n = split_buffer_inplace(paths, ':');
    char *p = paths;
    for(size_t i = 1; i <= n; i++) { // 1...n
        list_executables(i, p);
        // readover the field and the trailing '\0' to get to the next field
        p += strlen(p) + 1;
    }
    free(paths);
    return EXIT_SUCCESS;
}
```


1 Übersicht

In diesem Praktikum setzen Sie das Script vom Praktikum P01 in C um.

2 Lernziele

In diesem Praktikum möchten wir Sie mit der Handhabung von C Standard Library Funktionen und POSIX I/O Funktionen bekanntmachen.

- Sie können Memory allozieren und freigeben.
- Sie können in einem Buffer mit Standard Funktionen nach Zeichen suchen und diese in-place Ersetzen.
- Sie können mit POSIX Funktionen fragen, ob ein File ein Directory oder ein reguläres File ist und ob es Ausführungs-Berechtigung hat.
- Sie können den Inhalt eines Directories nach ausführbaren regulären Files durchsuchen.

- Sie wissen wie Sie die notwendigen Informationen über die genaue Anwendung von Library Funktionen über **`man`** Pages erhalten können

3 Original Script

In P01 haben Sie dieses Script analysiert und beschrieben, was es macht – und natürlich auch ausgeführt.

```
 1    #!/bin/bash
 2
 3    # produces a tabular output of all executables as found over the $PATH environment variable
 4    # - output format: [1-based index of $PATH entry]:[$PATH entry]:[name of the executable]
 5    #   - e.g. 6:/bin:bash
 6    # - the first argument (if given) is used as alternative to $PATH (e.g. for testing purposes)
 7    # usage: ./get-exec-list-arg.sh            # examines $PATH
 8    # usage: ./get-exec-list-arg.sh "$PATH"    # equivalent to the above call
 9    # usage: ./get-exec-list-arg.sh ".:~/bin"  # examines the current directory (.) and ~/bin
10
11    # argument handling
12    path="$1"
13    [ -n "$path" ] || path="$PATH"
14
15    # input-field-separator: tells the shell to split in the 'for' loop the $var by ":"
16    IFS=":"
17
18    for p in $path
19    do
20        i=$((i+1))
21        [ -n "$p" ] || p="."
22        if [ -d "$p" ] && [ -x "$p" ]
23        then
24            find -L "$p" -maxdepth 1 -type f -executable -printf "$i:%h:%f\n" 2>/dev/null
25        fi
26    done
```

4 Aufgabe

Sie werden das obige Script so in C umsetzen, dass es für denselben Input denselben Output wie das Script generiert.

Wenn Sie kein Argument angeben, dann wird die PATH Environment Variable als Ausgangspunkt genommen, ansonsten das erste Argument.

Der so erhaltene String wird an den ‘:’ in einzelne Felder aufgeteilt, wobei auch leere Felder entstehen können.

Jedes Feld wird als Directory Pfad angenommen. Für ein leeres Feld soll zur weiteren Verarbeitung ein Punkt (für das aktuelle Arbeits-Directory) genommen werden.

Für jeden solchen Pfad soll geprüft werden, ob es ein Directory ist und Ausführungsrechte hat.

Falls das so ist, soll jedes reguläre File in diesem Directory, wenn es Ausführungsrechte hat, gleich ausgegeben werden wie im Script.

4.1 Vorgegeben

Es ist ein Rahmenprojekt auf Git verfügbar unter dem Namen

`P08_Auflisten_aller_PATH_Executables`

Sie können das **`src/main.c`** File als Basis nehmen, oder Sie können alles von Grund auf selber schreiben.

Führen Sie die Tests und implementieren Sie die Funktionalität, bis die Tests erfolgreich durchlaufen.

Als Probe können Sie Ihr Programm und das Script von P01 nacheinander ausführen und dabei den Standard Output in ein File schreiben. Diese beiden Files können sie mit dem Shell Command **`diff`** vergleichen. Es sollte keine Unterschiede geben.

4.2 Zu verwendende Funktionen

In Ihrer Implementierung sollen Sie folgende Funktionen verwenden.

Wichtig: Immer Error Handling mit implementieren.

Funktion	Verwendung
`getenv`	**`PATH`** Variable lesen
`malloc`	dynamisch Speicher allozieren
`free`	dynamischen Speicher freigeben
`strchr`	Nach Zeichen suchen
`S_ISDIR`	Abfrage of ein mittels **`stat()`** angesprochenes File ein Directory ist
`S_ISREG`	Abfrage of ein mittels **`stat()`** angesprochenes File ein reguläres File ist
`stat`	Abfrage von File Attributen
`access`	Abfrage ob ein File Ausführungsrechte hat
`opendir`	Directory öffnen zum traversieren
`closedir`	Directory Zugriff abschliessen
`readdir`	traversieren des Directories – sequentiell Directory Einträge lesen

4.3 Tipps

Um einen String zu kreieren der Form «dir» «/» «Name» «\0» können Sie die Funktion **`sprintf`** verwenden. Dabei kenn es hilfreich sein, den Buffer zuerst dynamisch zu allozieren, dann via **`sprintf`** den Pfad String zusammenbauen, verwenden und schliesslich wieder frei zu geben.

Wenn diese Aufgabe erfolgreich umgesetzt ist, laufen die Tests ohne Fehler durch.

5 Bewertung

Die gegebenenfalls gestellten Theorieaufgaben und der funktionierende Programmcode müssen der Praktikumsbetreuung gezeigt werden. Die Lösungen müssen mündlich erklärt werden.

Aufgabe	**Kriterium**	**Punkte**
	Sie können das funktionierende Programm inklusive funktionierende Tests demonstrieren und erklären.	
1	die Aufgabe ist umgesetzt und sie können Ihre Implementation erklären.	8