Operating Systems

Exercise C: Arrays and Strings

1. ROT47

To get a basic understanding of strings in C, we are going to implement a version of the caesar cipher, the ROT47 algorithm.

Caesar ciphers are one of the older ciphers known to mankind. They simply shift each letter in a plain text by a number (the key). This produces the cipher. The decryption process simply reverses this procedure.

- T Create a file “rot47.c”.

We are using the same program for both encryption and decryption. ROT47 allows us to use the same algorithm for both procedures.

- T Ask the user for his input. This could be either a plain text, or an encrypted text. With the “%ms” modifier you can leave the allocation of the necessary memory to scanf(). Don’t forget to free your memory afterwards.

The encryption/decryption process is pretty straight forward. Simply shift each letter of the input according to the operating mode of the program.

- T Create a function with the following prototype:

```
char rot47(char plaintext)
```

- T The alphabet used by the ROT47 cipher has 94 characters reaching from “!” to “~”. Use this boundaries to implement the shift operation. The key is always 47 (surprise).
  - When shifting, prevent the cipher from overflowing by using the % operator. What does it do in this context?
  - Don’t shift whitespace.
- T Do we need to bother about “\0”-terminating our string? Why?
- T Print the encrypted/decrypted text to the console.

Operating Systems

Hints

These functions may be helpful to use. Use the manpage to get further information:

- strlen()

Includes

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>  // strlen()
```

Operating Systems

2. String-Objects

You may have already noticed that strings in C have a huge disadvantage compared to the ones in Java, Javascript or any other programming language: They don’t actually exist. This is why we want to build them ourselves today.

T Declare a structure with the following content

- A pointer to the actual content of the string.
- Variables for the length of the string (in bytes) and the maximum capacity of the string (the memory block where the string is stored).

T Add the following typedef after your struct definition:

```
typedef struct <your struct name> string_t;
```

With it you can define string variables like this:

```
string_t blablubb;
```

Operating Systems

3. String manipulation

Sometimes we have to extract a certain part of a string, often after some delimiter. In this exercise we are going to extract a substring after a newline. We are going to need this implementation later in the course of the lecture.

- T Create a file “string_manipulation.c”.
- T Implement a function with the following prototype. Use the string-structure from the previous exercise by copying it to this file.

```
string_t string_after_empty_line(string_t original)
```

- T The function should extract a substring of the original string after a the delimiter "\r\n\r\n" (without the delimiter). Return the new substring.
- T Use the following string to test your function. Simply put it in a new string object. The substring should only contain the HTML-document starting with the DOCTYPE.

```
HTTP/1.1 200 OK\r\nContent-Length: 107\r\nDate: Mon, 24 Nov 2014 10:21:21
GMT\r\n\r\n<!DOCTYPE html><html><head></head><body></body></html>
```

Hints

These functions may be helpful to use. Use the manpage to get further information:

- strstr()
- strlen()

Includes

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>
```