

# Exercises in System Level Programming (SLP) – Summer Term 2024

---

## Exercise 9

Maximilian Ott

Lehrstuhl für Informatik 4  
Friedrich-Alexander-Universität Erlangen-Nürnberg



Chair in Distributed Systems  
and Operating Systems



FRIEDRICH-ALEXANDER  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG

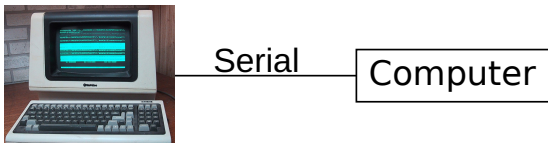
FACULTY OF ENGINEERING

**Linux**

---

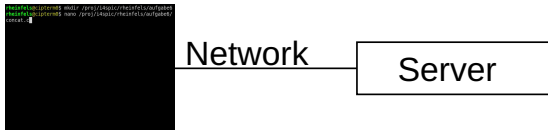


- When computers were bigger than today:



Televideo 925 (Public Domain: Wtshymanski @Wikipedia)

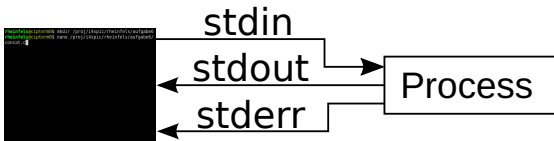
- When the internet was really slow:



- Colors, position jumps, etc. are indicated by special character sequences



- Three standard streams for in- and output



`stdin` Input

`stdout` Output

`stderr` Error message

- Standard behaviour
  - Inputs are received from the keyboard
  - Outputs & error messages appear on the screen



- Write `stdout` into a file

```
01 find . > directories.txt
```

- Use `stdout` as `stdin` for other programs

```
01 cat directories.txt | grep tmp | wc -l
```

- Advantage of `stderr`

⇒ Error messages are still displayed in the terminal

- Overview

> Write standard output `stdout` into file

>> Append standard output `stdout` to an existing file

2> Write error messages `stderr` into a file

< Read `stdin` from a file

| Use output of one command as input for another command



## ■ Change directory with `cd`

```
01 # absolute path to the directory
02 cd /proj/i4spic/<login>/aufgabeX/
03
04 # relative path to the directory
05 cd aufgabe5/
06
07 cd ~          # user directory ($HOME)
08 cd ..        # parent directory
```

## ■ List directory contents with `ls`

```
01 ls          # show files in current directory
02 ls -A       # also show hidden files
03 ls -lh      # show more meta data
```



- Copy file or directory with `cp`

```
01 # Copy file ampel.c from $HOME to the project directory
02 cp ~/ampel.c /proj/i4spic/xy42abcd/aufgabe5/ampel.c
03
04 # Copy directory aufgabe5/ from $HOME to the project directory
05 cp -r ~/aufgabe5/ /proj/i4spic/xy42abcd/
```

- (Permanently) Delete file or directory with `rm` (remove)

```
01 # Remove file test1.c inside the current directory
02 rm test1.c
03
04 # Remove subdirectory aufgabe1/ and all contained files
05 rm -r aufgabe1
```



- With a signal: CTRL-C (can be ignored by the program)
- Using another console: `killall concat` terminates all programs with the name “concat”
- Using the same console:
  - CTRL-Z stops the currently running process
  - `killall concat` then terminates all programs with the name `concat`
    - ⇒ Programs of other users shall never be terminated
  - `fg` continues the stopped process
- If nothing else works: `killall -9 concat`





The screenshot displays the SPiC IDE interface. On the left is a project tree for 'Project' containing a folder 'jy52coty' with subfolders 'aufgabe1' through 'aufgabe6' and files 'concat', 'concat.c', 'korrektur', and 'pub'. The main editor shows the content of 'concat.c':

```
concat.c
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main(int argc, char *argv[]) {
5     printf("Hello World\n");
6 }
```

Below the editor is the 'Atom Shell Commands' panel, showing the compilation command: `make -B trac` and `cc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3 -o trac trac.c -o trac` [Finished in 0.14 seconds].

At the bottom is a terminal window with the following session:

```
jy52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe5$ ls
concat.c
jy52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe5$ gcc -pedantic -Wall -Werror -O3 -std=c11 -D_XOPEN_SOURCE=700 -o
concat concat.c
jy52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe5$ ls
concat concat.c
jy52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe5$ ./concat
Hello World
jy52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe5$
```

- **Terminal:** opens a terminal and starts a shell
  - efficient interaction with the system
  - optionally full screen
- **Debug:** starts the debug mode
- **Make:** see next week



## ■ Compile program with GCC

```
01 gcc -pedantic -Wall -Werror -O3 -std=c11 -D_XOPEN_SOURCE=700 -o  
    → concat concat.c
```

- `gcc` calls the compiler (GNU Compiler Collection)
- `-pedantic` activates warnings (different to the C standard)
- `-Wall` activates warnings (typical errors, e.g.: `if(x = 7)`)
- `-Werror` makes warnings into errors
- `-O3` activates optimizations (level 3)
- `-std=c11` sets the used standard to C11
- `-D_XOPEN_SOURCE=700`  
adds certain POSIX extensions
- `-o concat` specifies the name of the output file (standard: `a.out`)
- `concat.c ...` file(s) that have to be compiled

## ■ Execute the program with `./concat`

## ■ All submitted assignments will be tested with these flags



- Compile the program with GCC  
(including debug symbols and without optimizations)

```
01 gcc -pedantic -Wall -Werror -O0 -std=c11 -D_XOPEN_SOURCE=700 -g -  
    → o concat concat.c
```

-O0 prevents the compiler from optimizing the program  
-g produces debug symbols in the executable file

⇒ enables the debugger to create references to the source file

- Hint: Arrow key ↑ iterates over previous commands

⇒ GCC command only has to be typed once



- Information about:
  - Memory leaks (`malloc(3)/free(3)`)
  - Invalid memory accesses
- Ideal for debugging segmentation faults (`SIGSEGV`)
- Calls:
  - `valgrind ./concat`
  - `valgrind --leak-check=full --show-reachable=yes`  
↳ `--track-origins=yes ./concat`
- The output is way more useful, if the analyzed binary was built with debug symbols



- Interface to the system reference manuals
- Divided into multiple sections
  - 1 Executable programs or shell commands
  - 2 System calls
  - 3 Library calls
  - 5 File formats and conventions (special data structures, etc.)
  - 7 Miscellaneous (e. g. terminal drivers, IP, ...)
- man pages are usually cited with the appropriate section:  
`printf(3)`

```
01 # man [section] term
02 man 3 printf
```

- Search for sections: `man -f term`
- Search man pages for a keyword: `man -k keyword`



- Trimmed (nicer) version of the man pages
- Only provide an overview and not a full specification
- Can be called from inside the SPiC-IDE (Hilfe-button when inside the Linux mode)
- Can be found on the website

<https://sys.cs.fau.de/lehre/ss24/slp/exercises/linux-libc-doc>

- Our overview does not replace the man pages
- In the exam: Printed man pages!

# Error Handling

---



- Errors can happen due to different reasons
  - System resources are completely exhausted
    - ⇒ `malloc(3)` fails
  - Invalid user inputs (e. g. non existent files)
    - ⇒ `fopen(3)` fails
  - Temporary errors (e. g. unavailable server)
    - ⇒ `connect(2)` fails





- Good software:
  - Detects the error
  - Handles error appropriately
  - Prints out a meaningful error message afterwards
  
- Can a program continue after an error occurred?

**Example 1:** Determining the hostname of an IP address to add both values to a log file

⇒ Add IP address to the log, program can continue

**Example 2:** Opening a file, that has to be copied, fails

⇒ Error handling: Copying impossible, terminate program

⇒ Or continue the copying process with the next file

⇒ Decision has to be made by the software developer



- Errors often occur in `libc` functions
  - Can (usually) be detected by the return value (man page)
  - Checking for errors is essential
  
- Error causes are usually written to `errno` (global variable)
  - Can be included with `errno.h`
  - Error codes are  $> 0$
  - Error codes for all possible errors (refer to `errno(3)`)
  
- Only evaluate `errno` if an error was signaled
  - Functions are allowed to modify `errno` arbitrarily
    - ⇒ `errno` can also be modified if no error occurred



- Print error codes:
  - `perror(3)`: Output on `stderr`
  - `strerror(3)`: Convert into error message (string)

## Example:

```
01 char *mem = malloc(...);
02
03 // Error case
04 if(NULL == mem) {
05     fprintf(stderr, "%s:%d: malloc failed with reason: %s\n",
06             __FILE__, __LINE__-5, strerror(errno));
07     //alternativ: perror("malloc");
08
09     exit(EXIT_FAILURE);
10 }
```



- Indicating an error via the return value is not always possible
- Return value EOF: Error case **or** End-Of-File

```
01 int c;  
02 while ((c=getchar()) != EOF) { ... }  
03 /* EOF or error? */
```

- Detection for I/O streams: `ferror(3)` bzw. `feof(3)`

```
01 int c;  
02 while ((c=getchar()) != EOF) { ... }  
03 /* EOF or error? */  
04 if(ferror(stdin)) {  
05     /* Error */  
06     ...  
07 }
```

## The Function main()

---



- Function `main()`: Entry point of a C program
- Signature depends on its usage:
  - AVR: Only one program
    - ⇒ `void main(void)`
  - Linux: Multiple programs
    - ⇒ `int main(void)`
    - ⇒ `int main(int argc, char *argv[])`
- Parameters and return value used for communication



- Command line arguments: Parameters for the program
- `main()` receives them as function parameters:
  - `argc`: Number of arguments
  - `argv`: Array of pointers to the arguments
    - ⇒ Array of strings
- First argument: program name



```
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 int main(int argc, char *argv[]) {
05     for(int i = 0; i < argc; ++i) {
06         printf("argv[%d]: %s\n", i, argv[i]);
07     }
08
09     return EXIT_SUCCESS;
10 }
```

```
01 $ ./commandline
02 argv[0]: ./commandline
03 $ ./commandline Hello world
04 argv[0]: ./commandline
05 argv[1]: Hello
06 argv[2]: world
```





- Return status: Information for the caller
- Usual codes:
  - EXIT\_SUCCESS: Execution succeeded
  - EXIT\_FAILURE: Error occurred



```
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 int main(int argc, char *argv[]) {
05     if(argc == 1) {
06         fprintf(stderr, "No parameters given!\n");
07         return EXIT_FAILURE;
08     }
09
10     // [...]
11
12     return EXIT_SUCCESS;
13 }
```

```
01 $ ./exitcode
02 No parameters given!
03 $ echo $?
04 1
05 $ ./exitcode Hello world
06 $ echo $?
07 0
```

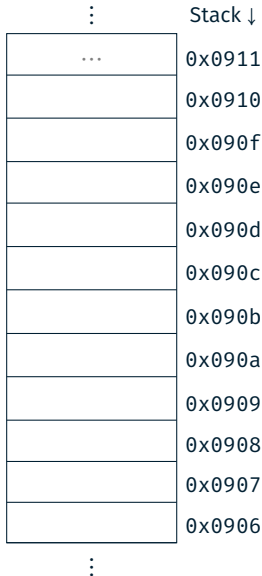
## C Strings in Detail

---



- char: Single character (e.g. 'a')
- String: Array of chars (e.g. "Hello")
- C: Last char of a string: '\0'  
⇒ Memory requirement:  $\text{strlen}(s) + 1$

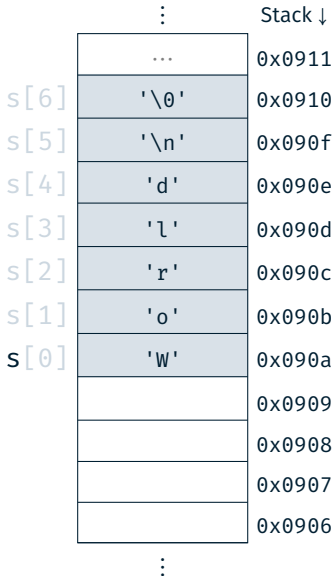
```
01 char s[] = "World\n";  
02 char c = s[0];  
03 c = s[4];  
04 char *s2 = s + 2;  
05 c = s2[1];
```





- char: Single character (e.g. 'a')
- String: Array of chars (e.g. "Hello")
- C: Last char of a string: '\0'  
⇒ Memory requirement:  $\text{strlen}(s) + 1$

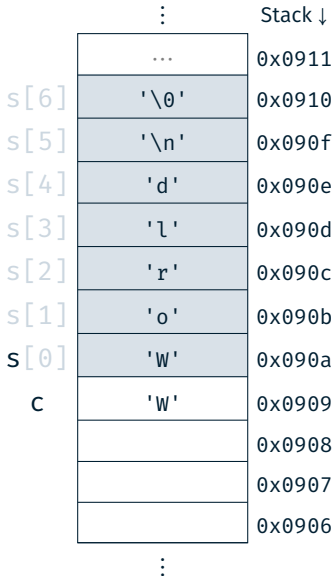
```
01 char s[] = "World\n";  
02 char c = s[0];  
03 c = s[4];  
04 char *s2 = s + 2;  
05 c = s2[1];
```





- char: Single character (e.g. 'a')
- String: Array of chars (e.g. "Hello")
- C: Last char of a string: '\0'  
⇒ Memory requirement:  $\text{strlen}(s) + 1$

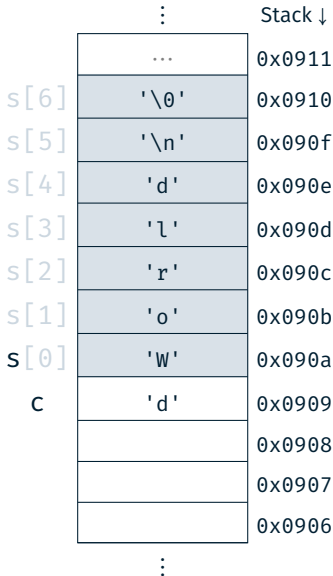
```
01 char s[] = "World\n";  
02 char c = s[0];  
03 c = s[4];  
04 char *s2 = s + 2;  
05 c = s2[1];
```





- char: Single character (e.g. 'a')
- String: Array of chars (e.g. "Hello")
- C: Last char of a string: '\0'  
⇒ Memory requirement:  $\text{strlen}(s) + 1$

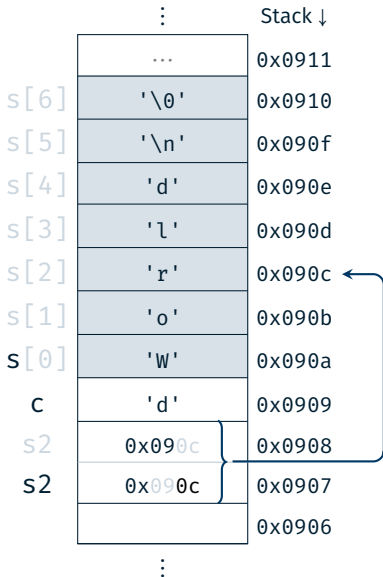
```
01 char s[] = "World\n";  
02 char c = s[0];  
03 c = s[4];  
04 char *s2 = s + 2;  
05 c = s2[1];
```





- char: Single character (e.g. 'a')
- String: Array of chars (e.g. "Hello")
- C: Last char of a string: '\0'  
⇒ Memory requirement:  $\text{strlen}(s) + 1$

```
01 char s[] = "World\n";
02 char c = s[0];
03 c = s[4];
04 char *s2 = s + 2;
05 c = s2[1];
```

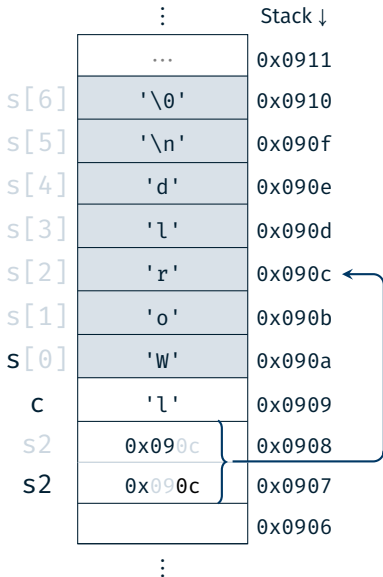






- char: Single character (e.g. 'a')
- String: Array of chars (e.g. "Hello")
- C: Last char of a string: '\0'  
⇒ Memory requirement:  $\text{strlen}(s) + 1$

```
01 char s[] = "World\n";  
02 char c = s[0];  
03 c = s[4];  
04 char *s2 = s + 2;  
05 c = s2[1];
```





- `size_t strlen(const char *s)`
  - Determine the length of a string `s` (without trailing NULL character)
- `char *strcpy(char *dest, const char *src)`
  - Copy a string `src` into a buffer `dest` (including NULL character)
  - Caution: Buffer overflow ( $\Rightarrow$  `strncpy(3)`)
- `char *strcat(char *dest, const char *src)`
  - Concatenate a string `src` after an existing string inside the buffer `dest` (including NULL character)
  - Caution: Buffer overflow ( $\Rightarrow$  `strncat(3)`)
- Documentation: `strlen(3)`, `strcpy(3)`, `strcat(3)`



```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include <string.h>
04
05 int main(void) {
06     const char *hello = "Hello";
07     const char *spic = "SPiC";
08
09     char altered_string[11]; // Space for "Hello SPiC"
10
11     strcpy(altered_string, hello); // "Hello"
12     strcat(altered_string, " "); // "Hello "
13     strcat(altered_string, spic); // "Hello SPiC"
14     strlen(altered_string); // -> 10
15
16     return EXIT_SUCCESS;
17 }
```

## Assignment: concat

---



- Concatenate the passed command line arguments into a single string and output of this string
- Procedure:
  - determine the required length
  - allocate the buffer dynamically
  - fill the buffer step by step
  - output the string on stdout
  - free the dynamically allocated buffer
- Re-implement the string library functions (from `string.h`):
- Important: identical behaviour (even in case of an error)

```
01  size_t str_len(const char *s)
02  char *str_cpy(char *dest, const char *src)
03  char *str_cat(char *dest, const char *src)
```



- `malloc(3)` allocates memory on the heap
  - reserves a minimum of `size` bytes of memory
  - returns a pointer to the start of the allocated memory
  - can potentially return an error
- `free(3)` frees the allocated memory again

```
01 char* s = (char *) malloc(...);
02 if(s == NULL) {
03     perror("malloc");
04     exit(EXIT_FAILURE);
05 }
06
07 // [...]
08
09 free(s);
```

# Hands-on: Buffer Overflow

---



- Program secured with a password

```
01 # Usage: ./print_exam <password>
02 ./print_exam spic
03 Correct Password
04 Printing exam...
```





- Program secured with a password

```
01 # Usage: ./print_exam <password>
02 ./print_exam spic
03 Correct Password
04 Printing exam...
```

- Unchecked user inputs  $\Rightarrow$  buffer overflow

```
01 long check_password(const char *password) {
02     char buff[8];
03     long pass = 0;
04
05     strcpy(buff, password);
06     if(strcmp(buff, "spic") == 0) {
07         pass = 1;
08     }
09     return pass;
10 }
```



- Program secured with a password

```
01 # Usage: ./print_exam <password>
02 ./print_exam spic
03 Correct Password
04 Printing exam...
```

- Unchecked user inputs  $\Rightarrow$  buffer overflow

```
01 long check_password(const char *password) {
02     char buff[8];
03     long pass = 0;
04
05     strcpy(buff, password);
06     if(strcmp(buff, "spic") == 0) {
07         pass = 1;
08     }
09     return pass;
10 }
```



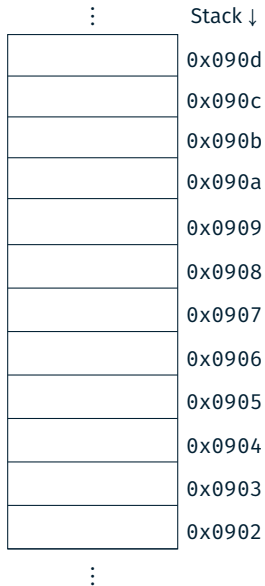
```
01 long check_password(const char *password) {
02     char buff[8];
03     long pass = 0;
04
05     strcpy(buff, password);
06     if(strcmp(buff, "spic") == 0) {
07         pass = 1;
08     }
09     return pass;
10 }
```

## ■ Possible solutions

- Check the user input
- Allocate the buffer dynamically
- Use of secure library functions ⇒ z.B. strncpy(3)



```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```





```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```

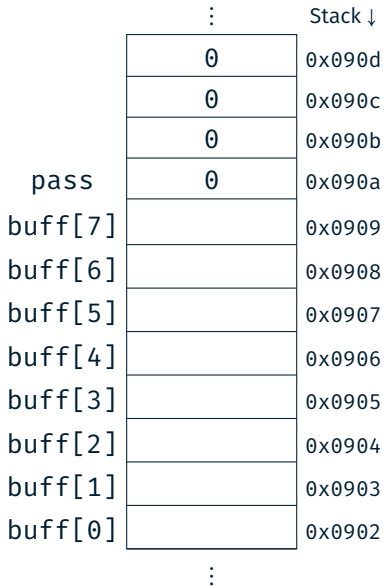
pass

:	Stack ↓
0	0x090d
0	0x090c
0	0x090b
0	0x090a
	0x0909
	0x0908
	0x0907
	0x0906
	0x0905
	0x0904
	0x0903
	0x0902

:



```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```





```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0	0x090b
pass	0	0x090a
buff[7]		0x0909
buff[6]		0x0908
buff[5]		0x0907
buff[4]	0 ('\0')	0x0906
buff[3]	99 ('c')	0x0905
buff[2]	105 ('i')	0x0904
buff[1]	112 ('p')	0x0903
buff[0]	115 ('s')	0x0902
	:	



```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0	0x090b
pass	0	0x090a
buff[7]		0x0909
buff[6]		0x0908
buff[5]		0x0907
buff[4]	0 ('\0')	0x0906
buff[3]	99 ('c')	0x0905
buff[2]	105 ('i')	0x0904
buff[1]	112 ('p')	0x0903
buff[0]	115 ('s')	0x0902
	:	





```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0	0x090b
pass	0	0x090a
buff[7]		0x0909
buff[6]		0x0908
buff[5]		0x0907
buff[4]	0 ('\0')	0x0906
buff[3]	99 ('c')	0x0905
buff[2]	105 ('i')	0x0904
buff[1]	112 ('p')	0x0903
buff[0]	115 ('s')	0x0902
	:	



```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0	0x090b
pass	1	0x090a
buff[7]		0x0909
buff[6]		0x0908
buff[5]		0x0907
buff[4]	0 ('\0')	0x0906
buff[3]	99 ('c')	0x0905
buff[2]	105 ('i')	0x0904
buff[1]	112 ('p')	0x0903
buff[0]	115 ('s')	0x0902
	:	

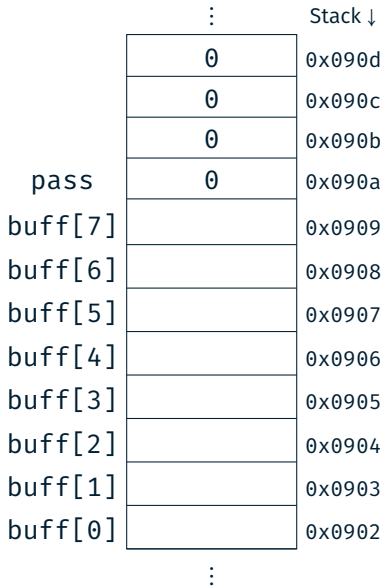


```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass; // pass = 1
13             // --> true
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0	0x090b
pass	1	0x090a
buff[7]		0x0909
buff[6]		0x0908
buff[5]		0x0907
buff[4]	0 ('\0')	0x0906
buff[3]	99 ('c')	0x0905
buff[2]	105 ('i')	0x0904
buff[1]	112 ('p')	0x0903
buff[0]	115 ('s')	0x0902
	:	



```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```





```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0	0x090b
pass	0	0x090a
buff[7]		0x0909
buff[6]		0x0908
buff[5]		0x0907
buff[4]		0x0906
buff[3]	0 ('\0')	0x0905
buff[2]	111 ('o')	0x0904
buff[1]	111 ('o')	0x0903
buff[0]	102 ('f')	0x0902
	:	



```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0	0x090b
pass	0	0x090a
buff[7]		0x0909
buff[6]		0x0908
buff[5]		0x0907
buff[4]		0x0906
buff[3]	0 ('\0')	0x0905
buff[2]	111 ('o')	0x0904
buff[1]	111 ('o')	0x0903
buff[0]	102 ('f')	0x0902
	:	



```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0	0x090b
pass	0	0x090a
buff[7]		0x0909
buff[6]		0x0908
buff[5]		0x0907
buff[4]		0x0906
buff[3]	0 ('\0')	0x0905
buff[2]	111 ('o')	0x0904
buff[1]	111 ('o')	0x0903
buff[0]	102 ('f')	0x0902
	:	



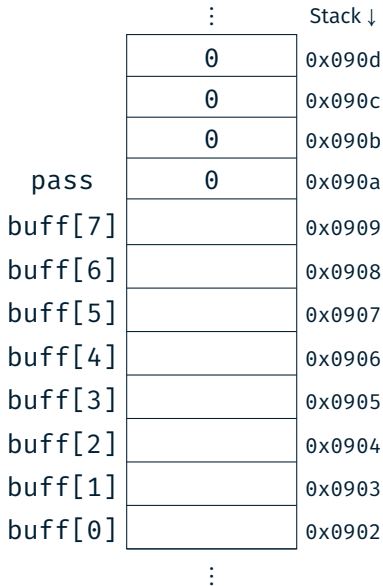
```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass; // pass = 0
13             // --> false
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0	0x090b
pass	0	0x090a
buff[7]		0x0909
buff[6]		0x0908
buff[5]		0x0907
buff[4]		0x0906
buff[3]	0 ('\0')	0x0905
buff[2]	111 ('o')	0x0904
buff[1]	111 ('o')	0x0903
buff[0]	102 ('f')	0x0902
	:	





```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```





```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0 ('\\0')	0x090b
pass	65 ('A')	0x090a
buff[7]	65 ('A')	0x0909
buff[6]	65 ('A')	0x0908
buff[5]	65 ('A')	0x0907
buff[4]	65 ('A')	0x0906
buff[3]	65 ('A')	0x0905
buff[2]	65 ('A')	0x0904
buff[1]	65 ('A')	0x0903
buff[0]	65 ('A')	0x0902
	:	



```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0 ('\\0')	0x090b
pass	65 ('A')	0x090a
buff[7]	65 ('A')	0x0909
buff[6]	65 ('A')	0x0908
buff[5]	65 ('A')	0x0907
buff[4]	65 ('A')	0x0906
buff[3]	65 ('A')	0x0905
buff[2]	65 ('A')	0x0904
buff[1]	65 ('A')	0x0903
buff[0]	65 ('A')	0x0902
	:	



```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass;
13
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0 ('\0')	0x090b
pass	65 ('A')	0x090a
buff[7]	65 ('A')	0x0909
buff[6]	65 ('A')	0x0908
buff[5]	65 ('A')	0x0907
buff[4]	65 ('A')	0x0906
buff[3]	65 ('A')	0x0905
buff[2]	65 ('A')	0x0904
buff[1]	65 ('A')	0x0903
buff[0]	65 ('A')	0x0902
	:	



```
01 long pass = 0;
02 char buff[8];
03 strcpy(buff, password);
04
05 if(strcmp(buff, "spic")) {
06     printf("Wrong Pass.\n");
07 } else {
08     printf("Correct Pass.\n");
09     pass = 1;
10 }
11
12 return pass; // pass = 65
13             // --> true
```

	:	Stack ↓
	0	0x090d
	0	0x090c
	0 ('\\0')	0x090b
pass	65 ('A')	0x090a
buff[7]	65 ('A')	0x0909
buff[6]	65 ('A')	0x0908
buff[5]	65 ('A')	0x0907
buff[4]	65 ('A')	0x0906
buff[3]	65 ('A')	0x0905
buff[2]	65 ('A')	0x0904
buff[1]	65 ('A')	0x0903
buff[0]	65 ('A')	0x0902
	:	

# Hands-on: Linux, GCC & Valgrind

Screenecast: <https://www.video.uni-erlangen.de/clip/id/18667>



- *Only online!*
- Goals:
  - Use SPiC IDE for Linux
  - Compile program from the command line
  - Practice the use of valgrind