

## Praktikumsaufgaben 1

|                     |  |
|---------------------|--|
| Gruppe (1,2 oder 3) |  |
| Name, Vorname       |  |
| Name, Vorname       |  |

|             | Datum                      | Abnahme |
|-------------|----------------------------|---------|
| Aufgabe 1.1 |                            |         |
| Aufgabe 1.2 |                            |         |
| Aufgabe 1.3 | wird im Praktikum gestellt |         |

### Hinweise zu den Praktikumsaufgaben:

- Bitte bearbeiten Sie die nachfolgenden Aufgaben für das 1.Praktikum in Ihrer Zweiergruppe. Sie können die Aufgabe soweit wie möglich zu Hause vorbereiten.
- Die Aufgabe 1.3 wird als Präsenzaufgabe im Praktikum verteilt.
- Für eine **Abnahme des Praktikums** sind alle nachfolgenden Punkte zu erfüllen.

#### Abgabe des Praktikumsberichtes

- Deckblatt: Aufgabenblatt
- kurze handschriftliche Beschreibung des Programms (Aufbau, Ablauf, Funktionen)
- Quellcode-Ausdruck
- Ergebnisausdruck

#### Vorstellung der Lösung der Aufgabe

- etwa 10-15 Minuten zur Erläuterung des Programms
- Vorführung des Programms
- Fragen/Antworten

#### Abgabe des Quellcodes

Durch Kopieren in das vorbereitete Verzeichnis unter dem Link:

xxxx

Kennzeichnen Sie die Aufgabe mit Praktikumsnummer und Aufgabennummer, z.B. Mueller\_Schulze\_Aufgabe1-2.c.

- Die Abnahme der Praktikumsaufgaben erfolgt im Praktikum.
- Kommentieren Sie ausreichend und verwenden Sie die **Programmierrichtlinien!**



**Hochschule für Angewandte  
Wissenschaften Hamburg**

*Hamburg University of Applied Sciences*

# **Praktikumsbericht**

PRP2-1

**Fabian Liebold, Malte Müller**

20. Oktober 2018

# Aufgabe 1: Pointer

## 1.a

Stellen Sie dar, welche Ausgabe das nachfolgend dargestellte Programm erzeugt.

```
1 H
2 e
3 l
4 l
5 o
6 Erster Wert von i war 5! Korrekt ?
7 H
8 I
9 J
10 K
11 L
12 Erster Wert von i war 0! Korrekt ?
```

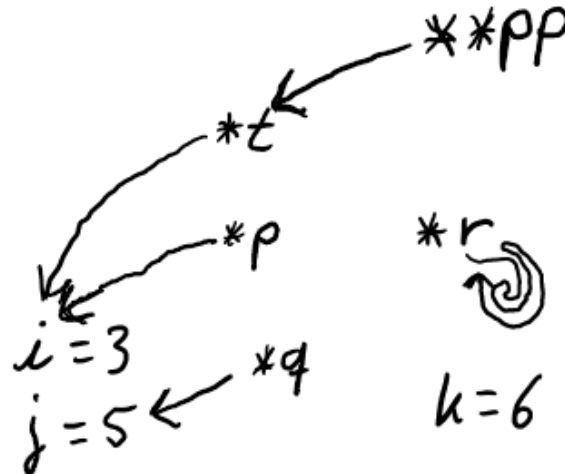
## 1.b

Geben Sie weiterhin die Hexadezimaldarstellung der Adressen sowie bei Zeigern den Inhalt, auf den er zeigt, an. Erweitern Sie hierfür das Programm durch printf-Anweisungen oder verwenden Sie den Debugger.

```
1 Adresse i: 010FFCE4
2 Wert i: 5
3
4 Adresse first_i: 010FFCCC
5 Wert first_i: 0
6
7 Adresse first_i_ptr: 010FFCD8
8 Wert first_i_ptr: 010FFCE4
9 Wert von Adresse 010FFCE4: 5
10
11 Adresse onechar: 010FFC4B
12 Wert onechar: M
13
14 Adresse strptr: 010FFC54
15 Wert strptr: 010FFC65
16 Wert von Adresse 010FFC65: World!
```

## 2.a

Zeichnen Sie graphisch die Zeigerabhängigkeiten auf.



## 2.b

```
1 wert = (p == i);
```

Wert ist 0. Da bei p nicht der Wert, sondern die Adresse betrachtet wird sind nicht beide echt gleich.

```
1 wert = *p / *q;
```

Hier wird 3/5 gerechnet und nur in ganzen Zahlen dargestellt, was hier 0 ist. Falls kein Leerzeichen zwischen / und \*p steht, denkt das Programm, es wird auskommentiert

```
1 wert = *p / *q + 3;
```

Hier wird zu dem Ergebnis 0 von der letzten Berechnung eine 3 hinzuaddiert, deshalb ist das Resultat 3.

```
1 wert = **&p;
```

Wert ist 3. Da der Wert hinter der Adresse, auf die \*\*p zeigt, dargestellt wird.

```
1 wert = *(r = &k);
```

Wert ist 6. r wird auf die Adresse von k gesetzt und als Pointer zeigt r auf die Adresse von k mit dem Wert 6.

```
1 wert = *(r = &k) = *p**q;
```

Wert ist 15. Zuerst wird \*r auf die Adresse von k gerichtet, dann wird die Adresse von k auf den Wert 15 (von 3(p)\*5(q)) gesetzt und am Ende wird „wert“ diesem Wert gleichgesetzt.

```
1 wert = *p***pp;
```

Wert ist 9. Der Wert auf den p zeigt, wird mit dem Wert auf den q zeigt multipliziert und auf „wert“ gespeichert.

## Aufgabe 2: Schach

### (1) Aufbau eines Arrays, dass das Schachbrett und die Schachfiguren darstellen kann

Es wurde ein zweidimensionales char-Array (playground) initialisiert. Die Größe des Arrays ist über ein Makro (SIZE) definiert. In unserem Fall entspricht die Größe den Dimensionen eines Schachfeldes. Bei dem Feld A1 fängt das Array bei [0][0] an.

Weiße Felder werden durch ein Leerzeichen dargestellt, für schwarze wird der Ascii-Code 177 verwendet.

### (2) String der Schachnotation in einzelne Züge aufschlüsseln

Wurde mit **strtok** (string token) realisiert, durch Festlegen eines Trennzeichens (hier: „/“). Rückgabe ist ein Pointer, der auf die einzelnen gespaltenen Teile zeigt. In einer Schleife wird dieser Prozess wiederholt, bis alle Kommandos abgearbeitet wurden.

### (3) Automatisches Nachspielen der gegebenen Züge

Nachdem ein Kommando aus dem Partie-String extrahiert wurde, wird die Funktion **play-str** aufgerufen und die Länge des Kommandos abgefragt. Bei einer Länge von 5 ist keine Spielfigur angegeben und es muss sich um einen Bauern handeln. Bei einer Länge von 6 ist die Spielfigur festgelegt. Als nächstes wird überprüft, ob es sich um einen Setzzug oder Schlagzug handelt. Die Stelle des ausschlaggebenden Characters ist durch die bereits abgeschlossene Abfrage zur Länge des Kommandos eindeutig.

Durch die counter-Variable, die in jedem Schleifendurchlauf erhöht wird, kann überprüft werden, welcher Spieler gerade am Zug ist. Mit der Modulo-Operation wird überprüft, ob die Spielfigur als Groß- oder Kleinbuchstabe in das playground-Array gespeichert werden muss. Anschließend wird die Funktion **update-playground** aufgerufen und das Spielfigur wird im Array auf die Zielposition gesetzt. Das Startfeld wird entweder durch ein weißes oder schwarzes Feld ersetzt.

### (4) Darstellung des veränderten Schachbrettes auf dem Bildschirm

Ausgegeben wird das Spielfeld mit einer einfachen for-Schleife in der Funktion **print-playground**.

### (5,6) Benutzer soll das Spiel weiterspielen

Sobald der Partie-String vollständig gespielt wurde, wird in der main-Funktion gefragt ob der Spieler weiterspielen möchte oder nicht. Dies kann mit einer einfachen Eingabe von „j“ bestätigt und mit „n“ verneint werden. Falls die Frage mit einem „Nein“ beantwortet wurde wird das Programm beendet. Falls weitergespielt werden möchte, wird die Funktion **play-input** aufgerufen. Hier wird zunächst die Startposition und die Zielposition abgefragt. Bei der Eingabe von „00“ als Startposition wird das Spiel beendet. Als nächstes werden die x- und y-Koordinaten der Startposition in Integer umgewandelt und die Figur, welche sich auf der Startposition befindet herausgearbeitet. Anschließend wird die Funktion **update-playground** aufgerufen und das Spielfeld wie bereits in (3) nachgespielt und anschließend auf der Konsole mit der Funktion **print-playground** ausgegeben.

```

1  /*
2  PRP2-1 Aufgabe 1.2
3  Name: Malte Müller, Fabian Liebold
4  Date: 18.10.2018
5  */
6
7  #define _CRT_SECURE_NO_WARNINGS
8  #define MAX 100 //define max lenght of command string
9  #define SIZE 8 // length and width of playground
10
11 #include <stdio.h>
12 #include <string.h>
13
14 // Prototypes:
15 void init_playground(char playground[SIZE][SIZE]);
16 void print_playground(char playground[SIZE][SIZE]);
17 void play_str(char playground[SIZE][SIZE], char partie_copy[MAX]);
18 void update_playground(int counter, char playground[SIZE][SIZE], char figure,
19 char old_x, char old_y, char new_x, char new_y);
20 int mapping(char input);
21 void play_input(char playground[SIZE][SIZE]);
22
23 int main(void) {
24
25     char partie[MAX] =
26         "e2-e4/e7-e5/Sg1-f3/Sb8-c6/Lf1-c4/Lf8-c5/Sf3xe5/Sc6xe5/De1-e3/Lc5-e3/"; //
27     given partie string
28     char partie_copy[MAX];
29     strcpy(partie_copy, partie); // Create copy for strtok
30
31     //Print to console:
32     printf("*** Schach - Nachspielen einer Partie ***\n\n");
33     printf("Folgende Parite wird gespielt:\n");
34     printf("%s\n", partie);
35     printf("Die Zuege sind beginnend mit weiss abwechselt notiert.\nDie weissen
36     Figuren im Spielfeld mit Grossbuchstaben bezeichnet.\n");
37
38     //Initialize playground array and print it to console
39     char playground[SIZE][SIZE];
40     init_playground(playground);
41     print_playground(playground);
42
43     play_str(playground, partie_copy); // play the commands from the string
44
45     char flag = 'N';
46     printf("Wollen Sie die Partie weiterspielen? [J/N]:");
47     scanf("%c", &flag);
48     if(flag == 'j' || flag == 'J') play_input(playground); // check if user wants
49     to play further
50
51     printf("\nSpiel beendet.\n");
52
53     system("PAUSE");
54     return 1;
55 }
56
57 /*****
58 *****
59 Function initializes playground array.
60 Parameters:
61     char playground[SIZE][SIZE]
62 Returns:
63     void.
64 */
65 void init_playground(char playground[SIZE][SIZE]) {
66
67     playground[0][0] = 'T';
68     playground[0][1] = 'S';

```

```

69     playground[0][2] = 'L';
70     playground[0][3] = 'K';
71     playground[0][4] = 'D';
72     playground[0][5] = 'L';
73     playground[0][6] = 'S';
74     playground[0][7] = 'T';
75
76     playground[7][0] = 't';
77     playground[7][1] = 's';
78     playground[7][2] = 'l';
79     playground[7][3] = 'k';
80     playground[7][4] = 'd';
81     playground[7][5] = 'l';
82     playground[7][6] = 's';
83     playground[7][7] = 't';
84
85     for (int i = 0; i < SIZE; i++) {
86         playground[1][i] = 'B';
87     }
88
89     for (int i = 0; i < SIZE; i++) {
90         playground[6][i] = 'b';
91     }
92
93
94     for (int i = 2; i < 6; i++) {
95         for (int j = 0; j < SIZE; j++) {
96             if ((i % 2 == 0 && j % 2 == 0) || i % 2 == 1 && j % 2 == 1) {
97                 playground[i][j] = ' ';
98             }
99             else playground[i][j] = 177;
100         }
101     }
102 }
103
104 /*****
105 *****
106 Function prints the playground to the console.
107 Parameters:
108     char playground[SIZE][SIZE]
109 Returns:
110     void.
111 */
112 void print_playground(char playground[SIZE][SIZE]) {
113     printf("\nSpielbrett\n    |\n");
114     for (int i = 7; i >= 0; i--) {
115         printf("    %i | %c %c %c %c %c %c %c %c \n", i+1, playground[i][0],
116             playground[i][1], playground[i][2],
117             playground[i][3], playground[i][4], playground[i][5], playground[i][6],
118             playground[i][7]);
119     }
120     printf("----+-----\n    | a b c d e f g h\n\n");
121 }
122
123 /*****
124 *****
125 Function, which handles the commands of the given string.
126 Parameters:
127     char playground[SIZE][SIZE]
128     char partie_copy[MAX] - given command string.
129 Returns:
130     void.
131 */
132 void play_str(char playground[SIZE][SIZE], char partie_copy[MAX]) {
133     char delimiter[] = "/"; // set delimiter
134     char* token_ptr = strtok(partie_copy, delimiter); // Create pointer, which
135     points on the beginning of each word
136     char fig = ' ';
137
138     int counter = 1;
139     int play_flag = 1;

```

```

137
138 while (token_ptr != 0) { // While a command is available
139
140     system("PAUSE"); // Wait until the user presses 'return' to continue
141
142     printf("\n%i.Zug: %s\n", counter, token_ptr); // Print current command
143
144     if (strlen(token_ptr) == 5) { // Check the length of the command
145         if (token_ptr[2] == '-') { // differentiate if it's "Setzzug" or
            "Schlagzug"
146             printf("Dieser Zug ist ein Setzzug:\n");
147         }
148         else {
149             printf("Dieser Zug ist ein Schlagzug:\n");
150         }
151
152         if (counter % 2 == 1) fig = 'B';
153         else fig = 'b';
154
155         update_playground(counter, playground, fig, token_ptr[0], token_ptr[1],
            token_ptr[3], token_ptr[4]); // update playground
156     }
157     else if (strlen(token_ptr) == 6) {
158         if (token_ptr[3] == '-') {
159             printf("Dieser Zug ist ein Setzzug:\n");
160         }
161         else {
162             printf("Dieser Zug ist ein Schlagzug:\n");
163         }
164         if (counter % 2 == 1) fig = token_ptr[0];
165         else fig = token_ptr[0] + 32; // If counter is odd -> get small
            character by adding 32 to the ascii value
166
167         update_playground(counter, playground, fig, token_ptr[1], token_ptr[2],
            token_ptr[4], token_ptr[5]); // update playground
168     }
169
170     print_playground(playground); // Print updated playground-array to console
171
172     token_ptr = strtok(NULL, delimiter); // reset token pointer
173     counter++; // Increase counter
174 }
175 }
176
177 /*****
178 *****
179 Function updates the playground-array.
180 Parameters:
181     int counter - counts turns.
182     char playground[SIZE][SIZE]
183     char figure - figure, which should be moved.
184     char old_x - start position.
185     char old_y
186     char new_x - destination position.
187     char new_y
188 Returns:
189     void.
190 */
191 void update_playground(int counter, char playground[SIZE][SIZE], char figure,
    char old_x, char old_y, char new_x, char new_y) {
192
193     playground[(int)new_y - 49][mapping(new_x)] = figure; // Get the current
        figure of the start-position
194
195                                     //(-49 because Number is a character and
        has to be converted to int)
196
197     if ((mapping(old_x) % 2 == 0 && ((int)old_y - 49) % 2 == 0) ||
        (mapping(old_x) % 2 == 1 && ((int)old_y - 49) % 2 == 1)) { // check the
        pattern
198         playground[(int)old_y - 49][mapping(old_x)] = ' ';
199     }
200     else playground[(int)old_y - 49][mapping(old_x)] = 177;
201

```



```

202 }
203
204 /*****
*****
205 Function, which handles the commands of the given string.
206 Parameters:
207     char input - character, which should be mapped to corresponding int.
208 Returns:
209     int - corresponding int.
210 */
211 int mapping(char input) {
212     /*
213     Function maps the characters a to h to its corresponding numbers
214     */
215
216     int v = 0;
217
218     switch (input)
219     {
220     case 'a':
221         v = 0;
222         break;
223     case 'b':
224         v = 1;
225         break;
226     case 'c':
227         v = 2;
228         break;
229     case 'd':
230         v = 3;
231         break;
232     case 'e':
233         v = 4;
234         break;
235     case 'f':
236         v = 5;
237         break;
238     case 'g':
239         v = 6;
240         break;
241     case 'h':
242         v = 7;
243         break;
244     default:
245         break;
246     }
247
248     return v;
249 }
250
251 /*****
*****
252
253 Fuction handles user input.
254 Parameters:
255     char playground[SIZE][SIZE]
256 Returns:
257     void.
258 */
259 void play_input(char playground[SIZE][SIZE]) {
260     /*
261     Fuction handles user input.
262     */
263     char old_value[3]; // Create String of length of 3 "XX\0"
264     char new_value[3]; // Create String of length of 3 "XX\0"
265     int counter = 1;
266     char fig;
267     int x, y;
268
269     while (1) { // Create a loop
270
271         printf("Beenden der Partie durch Eingabe eines 00!\n");
272         printf("Eingabe Ausgangsfeld [SpalteZeile]: ");

```

```
273     scanf("%s", old_value);
274
275     if (old_value[0] == '0' && old_value[1] == '0') return; // Return to main,
    if user inputs "00"
276
277     printf("Eingabe Zielfeld [SpalteZeile]: ");
278     scanf("%s", new_value);
279
280     x = (int)old_value[1] - 49; // Get x value by converting the character to
    int and subtract 49 from ascii value
281     y = mapping(old_value[0]); // Get y value by mapping the character to its
    number
282
283     fig = playground[x][y]; //get selected figure
284
285     update_playground(counter, playground, fig, old_value[0],
286         old_value[1], new_value[0], new_value[1]); // update playground
287
288     print_playground(playground); // print playground
289
290     counter++; // Increase counter
291 }
292
293 }
```

```

1  /*
2  PRP2-1 Aufgabe 1.3
3  Name: Malte Müller, Fabian Liebold
4  Date: 18.10.2018
5  */
6
7  #define _CRT_SECURE_NO_WARNINGS
8  #include <stdio.h>
9
10
11 //Prototypen:
12 void eingabe(char* ptr);
13 void umwandeln(char* ptr);
14 void ausgabe(char* ptr);
15
16 char eingabe_v(char ptr);
17 char umwandeln_v(char ptr);
18 void ausgabe_v(char ptr);
19
20 int main(void) {
21     char zeichen = ' ';
22
23     // Aufgabe 1:
24     eingabe(&zeichen);
25     ausgabe(&zeichen);
26     umwandeln(&zeichen);
27     ausgabe(&zeichen);
28
29     fseek(stdin, 0, SEEK_END); // Puffer leeren
30
31     ausgabe_v(umwandeln_v(eingabe_v(zeichen)));
32
33     system("PAUSE");
34
35     return 0;
36 }
37
38 /*****
39 Eingabe des Buchstabens.
40 Parameter:
41     char *ptr - pointer which points on character.
42 Rückgabe:
43     void.
44 */
45 void eingabe(char* ptr) {
46     printf("Bitte geben Sie einen Buchstaben ein: ");
47     scanf("%c", ptr); // Abfrage des Zeichens
48 }
49
50 /*****
51 Umwandeln der Kleinbuchstaben in Großbuchstaben.
52 Parameter:
53     char *ptr - pointer which points on character.
54 Rückgabe:
55     void.
56 */
57 void umwandeln(char* ptr) {
58     if (*ptr < 123 && *ptr > 96) { // Falls KLeinbuchstabe -> Umwandeln in
        Großbuchstabe
59         printf("Umwandlung des Zeichens!\n");
60         *ptr = *ptr - 32;
61     }
62 }
63
64 /*****
65 Ausgabe des Buchstabens.
66 Parameter:
67     char *ptr - pointer which points on character.
68 Rückgabe:
69     void.

```

```

70  */
71  void ausgabe(char* ptr) {
72      printf("Das Zeichen ist: %c\n", *ptr);
73  }
74
75  //Aufgabe 2:
76  /*****
77  Eingabe des Buchstabens.
78  Parameter:
79      char zeichen - poiter which points on character.
80  Rückgabe:
81      char zeichen
82  */
83  char eingabe_v(char zeichen) {
84      printf("Bitte geben Sie einen Buchstaben ein: ");
85      scanf("%c", &zeichen); // Abfrage des Zeichens
86
87      ausgabe_v(zeichen);
88
89      return zeichen;
90  }
91
92  /*****
93  Umwandeln der Kleinbuchstaben in Großbuchstaben.
94  Parameter:
95      char zeichen - poiter which points on character.
96  Rückgabe:
97      char zeichen
98  */
99  char umwandeln_v(char zeichen) {
100      if (zeichen < 123 && zeichen > 96) {
101          printf("Umwandlung des Zeichens!\n");
102          zeichen = zeichen - 32;
103      }
104
105      return zeichen;
106  }
107
108  /*****
109  Ausgabe des Buchstabens.
110  Parameter:
111      char zeichen - poiter which points on character.
112  Rückgabe:
113      void.
114  */
115  void ausgabe_v(char zeichen) {
116      printf("Das Zeichen ist: %c\n", zeichen);
117  }

```