Softwaredokumentation

Matrikelnummer: 9026481

1. Kurze Zusammenfassung der Aufgabe:

Es soll das Spiel Master Mind programmiert werden. Dabei versucht der Spieler einen vom Computer zufällig generierten Code mit bis zu 10 Versuchen zu erraten. Zur Unterstützung bekommt der Spieler nach jedem Versuch eine Auswertung auf RWS (Richtiger Wert an richtiger Stelle) und RW (Richtiger Wert falsche Stelle).

2. Ansatz zur Umsetzung:

Für mich ist das Spiel Mastermind am einfachsten nachzuvollziehen, wenn ich eine Schlüssel-Schloss Analogie verwende. Dabei ist das "Schloss" der Code, den das Programm zufällig generiert. Der Spieler versucht dann dieses Schloss zu knacken und kann sich dafür bis zu 10 "Schlüssel" ausdenken. Das Schloss wird dafür in einem gleichnamigen Array mit der Größe 5 gespeichert welches zu Beginn mit 0 Initialisiert wird, aber dann sofort mit dem ZufallsCodeGenerator „befüllt“ wird.

Die 10 Schlüssel hingegen werden in dem mehrdimensionalen Array "Schlüsselbund" gespeichert. Dieser hat 10 Zeilen a 7 Spalten.

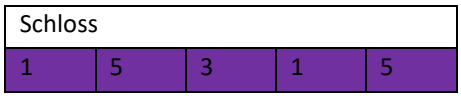
In die Zeilen werden also bis zu 10 "Schlüssel" des Spielers gespeichert (gelb). In blau ist die Bedeutung der jeweiligen Spalte geschrieben, dabei sind die ersten 5 Spalten für den 5-stelligen Schlüssel reserviert (hellgrün), während die letzten beiden Spalten für das Speichern der Auswertung der Schlüssel auf RWS/RW reserviert sind(dunkelgrün).

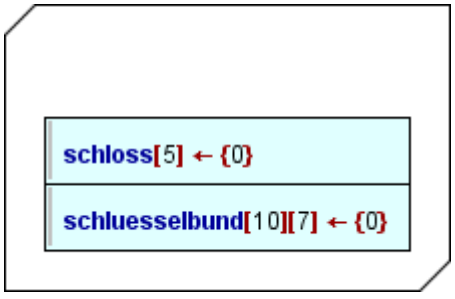
Schema für den Schlüsselbund und das Schloss mit ein paar beispielhaften Werten befüllt.

(Der grün markierte Bereich ist das tatsächliche Array, der Rest dient nur zur Veranschaulichung).
Zu Beginn des Programms steht in jeder Zelle eine 0.

Schlüsselbund

z\s	0.	1.	2.	3.	4.	RW	RWS
0.	1	2	3	4	5	3	0
1.	1	1	1	1	1	2	0
2.	2	2	2	2	2	0	0
3.	3	3	3	3	3	1	0
4.	4	4	4	4	4	0	0
5.	1	3	1	5	5	2	3
6.	1	5	3	1	5	5	0
7.	0	0	0	0	0	0	0
8.	0	0	0	0	0	0	0
9.	0	0	0	0	0	0	0





Beim Erstellen des Programms habe ich versucht, um das Programm so übersichtlich und modular wie möglich zu gestalten, für alle Relevanten Aspekte eine eigene Funktion zu schreiben. So bin ich am Ende auf 6 Funktionen gekommen (exkl. Main)

1. void zufalls_Code_Generator(int schloss[5]);
2. int einlesen(int z, int schluesselbund[10][7]);
3. void ausgeben(int z, int schluesselbund[10][7]);
4. void pruefen_RWS(int z, int schloss[5], int schluesselbund[10][7]);
5. void pruefen_RW(int z, int schloss[5], int schluesselbund[10][7]);
6. int spiel(int z, int schloss[5], int schluesselbund[10][7]);

Zu Beginn des Programms wird ein zufälliger Code erstellt. Dann werden bis zu 10 Runden gespielt. Die Funktion Spiel geht dabei immer eine Runde des Spiels durch, bestehend aus:

Spiel: einlesen→ pruefenRWS→ pruefenRW→ausgeben

Nach dem Einlesen des Schlüssels wird dieser also erst auf RWS geprüft, dann auf RW. Danach wird das Ergebnis ausgegeben. Wichtig ist hierbei, dass nicht nur die aktuelle Runde ausgegeben wird, sondern auch alle vorherigen, damit der Spieler einen guten Überblick über seine Tipps hat.

Herausforderungen beim Erstellen des Programms:

<u>Eingaben</u>:

Eine Eingabe von unerwünschten Zahlen durch den Spieler soll unterbunden werden, damit der Spieler sich keinen ungewollten Vorteil erschaffen kann (z.B. durch die Eingabe von Zahlen größer als 5 um sozusagen einen „Platzhalter“ zu haben der immer RW/RWS = 0 zur Folge hat).

Außerdem muss verhindert werden, dass Zahlen eingelesen werden, die das Programm zum Abstürzen bringen würden. Hier kommen vor allem sehr große Zahlen in Frage, oder sehr kleine (negative) und zusätzlich jedes Zeichen.

Die Logik, die das verwirklicht, steckt in der Einlesen Funktion.

Wichtig war es mir dabei allerdings auch, dass bei einer falschen Eingabe das Programm nicht beendet wird, sondern die aktuelle Runde einfach neugestartet wird, damit der Spielspaß nicht verloren geht, wenn man versehentlich eine falsche Eingabe tätigt und schon eine Menge Zeit in die Lösung des Codes gesteckt hat.

Das habe ich über den Rückgabewert der Funktion gelöst, der im Fehlerfall den Wert 2 zurückgibt, was in Main untersucht wird und ggf. den Rundenzähler der for-Schleife einfach um eins reduziert.

Prüfen auf RW:

Das schwierigste an dem ganzen Programm war für mich das Prüfen auf RW, da es doch wesentlich komplexer ist, als es auf den ersten Blick scheint. Der finale Lösungsalgorithmus erkennt ein RW an folgenden Merkmalen (weiter unten befindet sich ein kleines Beispiel):

1. schluesselbund[z][sKey]==schloss[sLock] (Blauer Pfeil)
   Hier wird geguckt, ob der Wert in der Stelle sLock im Schloss gleich dem Wert in der Stelle sKey des Schlüssels ist.
2. schluesselbund[z][sLock]!=schloss[sLock] (Schwarzer Pfeil)
   Hier wird geprüft, dass der Wert in der Stelle sLock im Schlüssel NICHT gleich dem Wert in der Stelle sLock im Schloss ist. (Da sonst ein RWS vorliegen würde).

3. schluesselbund[z][sKey]!=schloss[sKey] (Grüner Pfeil)
   Hier wird geprüft, dass der Wert in der Stelle sKey des Schlüssels NICHT gleich dem Wert in der Stelle sKey im Schloss ist. (auch hier würde es sich sonst um einen RWS handeln).

4. doppel[sLock]== 0 (Gelber Pfeil)
   Hier wird geguckt, dass an der Stelle sLock im „doppel“ Array eine 0 steht. Wenn dies nicht der Fall ist, bedeutet das, dass an der Stelle sLock schon ein RW gefunden wurde und es darf nicht zu Dopplungen kommen.

```
for(sKey=0; sKey<5; sKey++){
    for(sLock=0; sLock<5; sLock++){
```

Da die Schleife für den Schlüssel außen liegt wird erst die erste Zelle des Schlüssels mit den 5 Zellen des Schlosses verglichen bevor die 2. Stelle im Schlüssel untersucht wird.

In diesem Beispiel ist sKey=3 und sLock=1;

Schloss	1	5	5	1	5
Schlüssel(bund)	1	3	1	5	5
doppel (Start)	0	0	0	0	0
doppel (Ende)	0	1	0	0	0

In dem Beispiel ist die 1. Bedingung True, die 2. Bedingung True, die 3. Bedingung True und die 4. Ist auch True. Es handelt sich also um einen RW.

In Folge dessen wird der Zähler für RW um 1 erhöht, und im doppel Array ist an der Stelle sLock nun eine 1, d.h. die Stelle sLock=1 im Schloss kommt nicht mehr für einen RW infrage, wodurch Dopplungen ausgeschlossen werden. Die Untersuchung vom Schlüssel an der Stelle sKey=3 wird automatisch beendet sobald ein RW gefunden wird und als nächstes wird nun die Stelle sKey=4 im Schlüssel untersucht.

Konkretes Beispiel für die Sinnhaftigkeit des doppel Arrays

(in Grün das Schloss, darunter 2 Schlüssel).

1	3	3	2	5	RWS	RW
4	4	4	4	4	0	0
4	4	1	1	4	0	2/1

Aus dem ersten Tipp wissen wir, dass keine 4 im Schloss vorkommt und können diese also als „Platzhalter“ benutzen. Im nächsten Tipp geben wir zweimal eine 1 ein, an Stelle 2 und 3. Ohne den „doppel-Test“ würden wir bei der Auswertung 2 RW bekommen. Das würde den Verdacht nahelegen, dass auch 2 Einsen im Schloss vorkommen, was allerding nicht der Fall ist. Korrekt ist also bei der Auswertung 1 RW.

Dies wird erreicht durch das doppel Array, welches an der Stelle sLock=0 auf 1 gesetzt wird, sobald der erste RW mit der 1 an der Stelle sKey=2 gefunden wird und somit wird die 1 im Schloss an der Stelle sLock für weitere RW geblockt.

3. Ausblick

Optimierungsmöglichkeiten:

Um die Performance des Programms zu erhöhen könnte man die beiden Funktionen pruefenRW und pruefenRWS zusammen in eine pruefen Funktion tun, da in pruefenRW sowieso auch auf RWS geprüft wird (allerding nur um diese rauszufiltern). Das hat zur Folge, dass folgender Operation unnötiger Weise 2 mal ausgeführt wird:

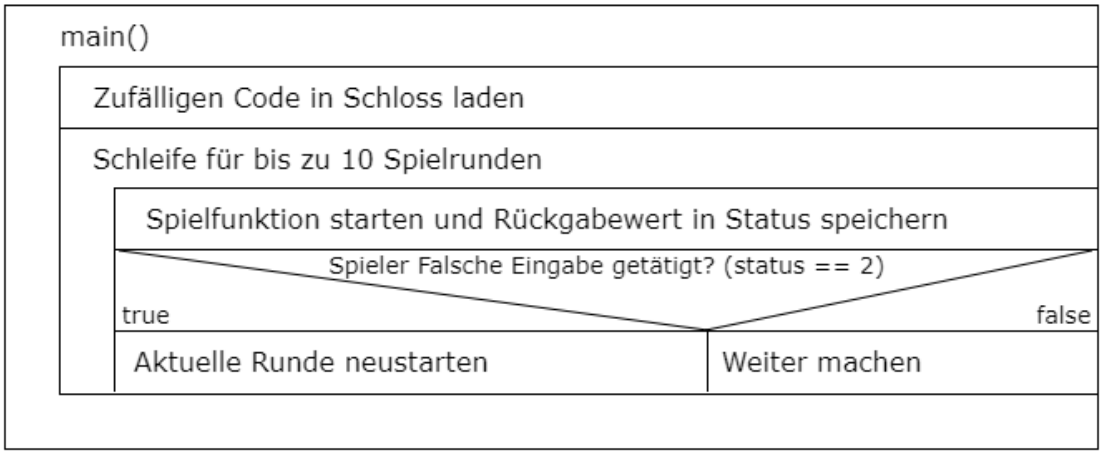
```
for(s=0; s<5; s++){
        if (schloss[s]==schluesselbund[z][s])
```

Ich habe dies nicht gemacht, da diese Operation ohnehin recht simpel ist und nicht wirklich viel Rechenleistung/-Zeit in Anspruch nimmt. Dafür habe ich ein etwas übersichtlicheres Programm, welches, gerade in Anbetracht des doch recht komplexen Aufbaus der pruefenRW Funktion, meiner Meinung nach wichtiger ist als die geringe Performance Verbesserung.

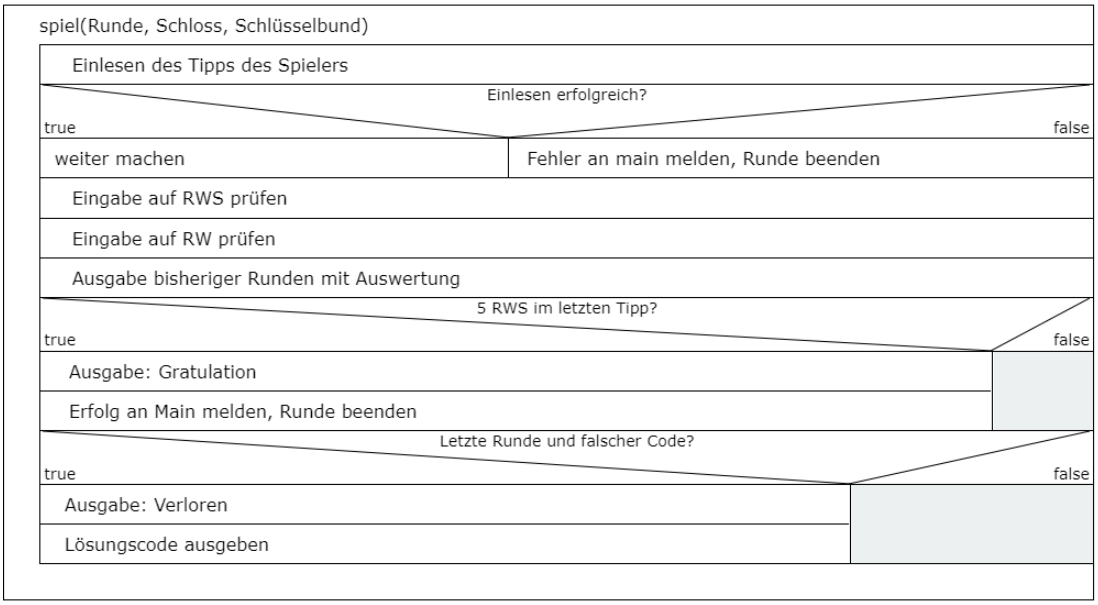
Eine weitere alternative Umstrukturierung des Programms könnte sein, die Spiel Funktion komplett zu löschen und die Syntax einfach in die Main Funktion zu schreiben. Wenn in der Einlesen Funktion ein Fehler passiert gibt diese eine 2 zurück an die Spiel Funktion welche Sie aufgerufen hat. Von dort aus wird diese Rückgabe dann an Main weitergegeben, erst dort wird dann der Fehler ausgewertet. Dieser Zwischenschritt ist leider eher unschön und könnte mit dem zusammenfügen von Main und Spiel gelöst werden, aber auch hier wäre dann die Übersichtlichkeit nicht so gut wie Sie jetzt ist und man hätte recht viel Syntax in der Main Funktion.

4. Struktogramm

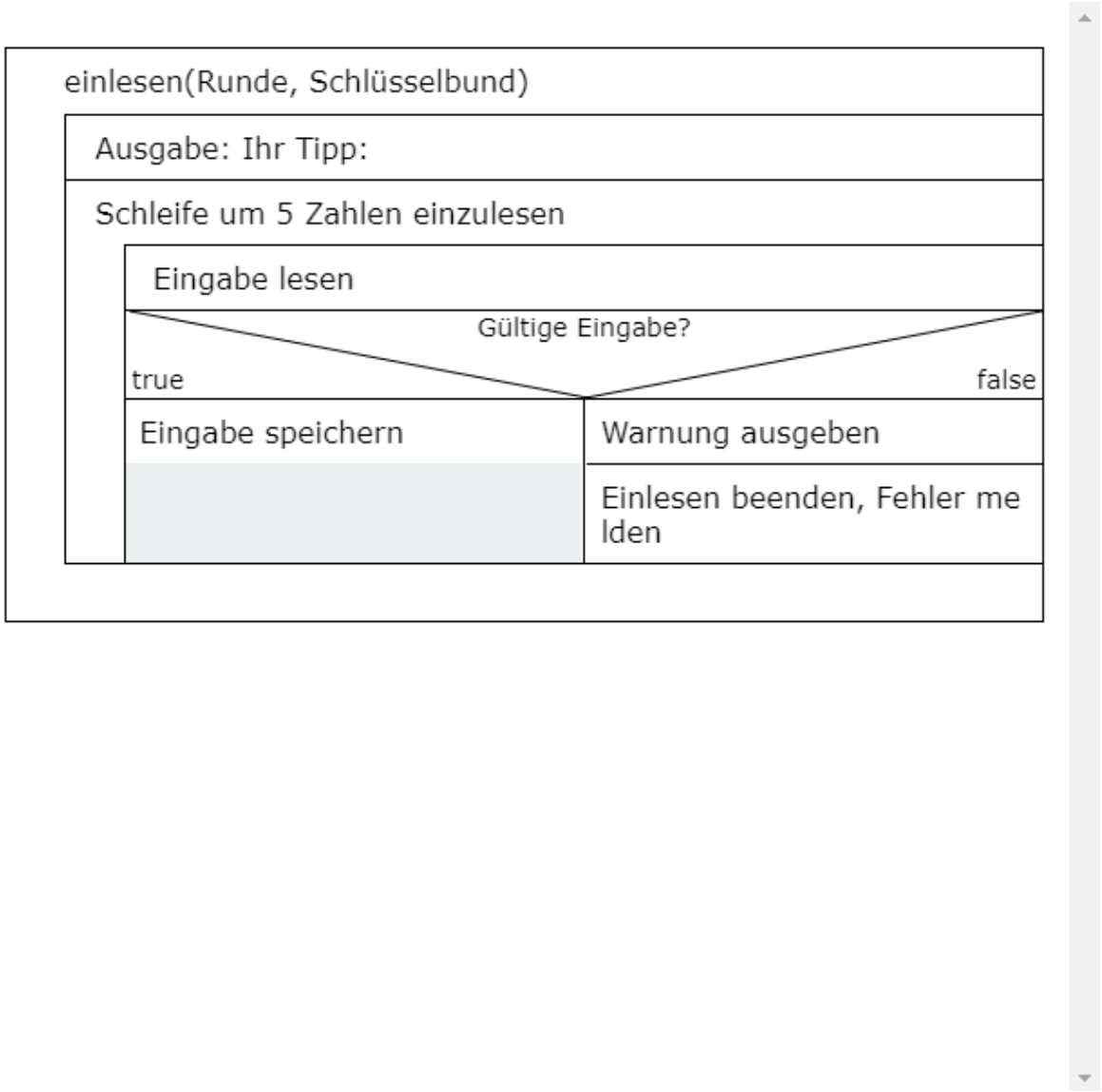
Main:



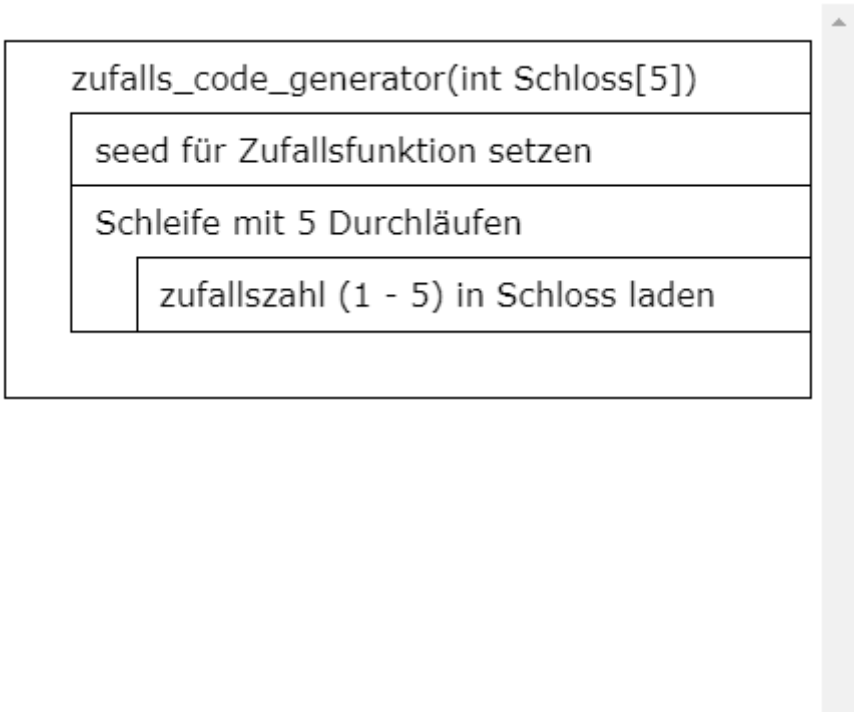
spiel:

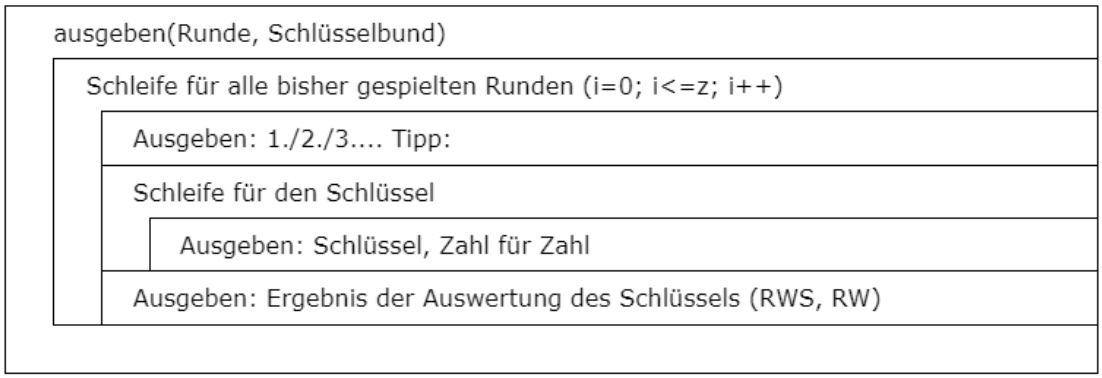


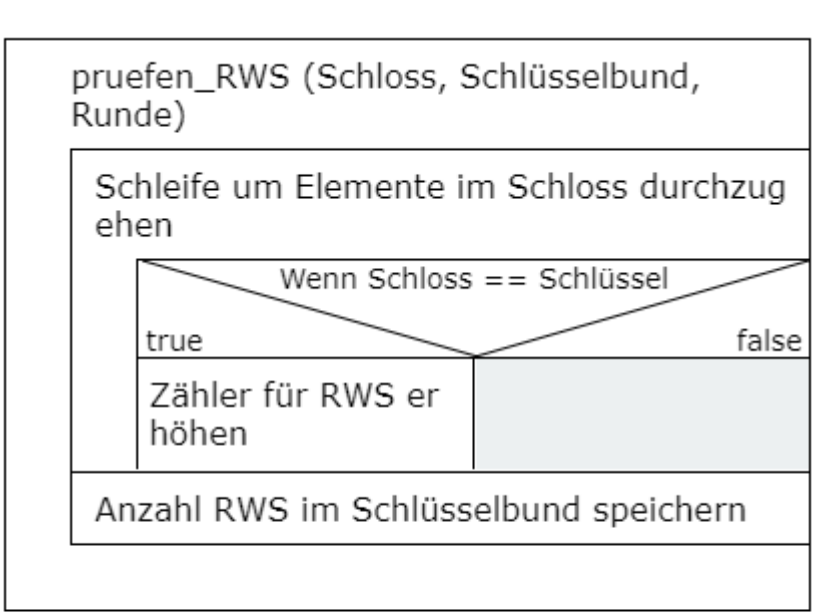
einlesen:

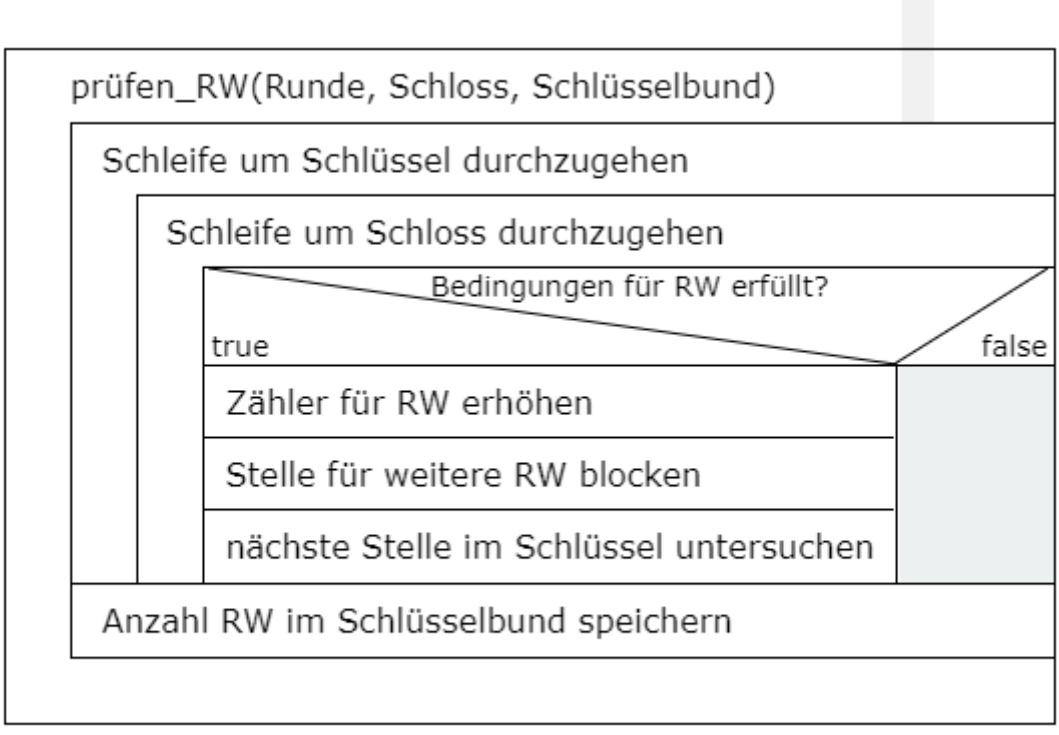


Zufallscodegenerator:




ausgeben(Runde, Schlüsselbund)
Schleife für alle bisher gespielten Runden (i=0; i<=z; i++)
Ausgeben: 1./2./3.... Tipp:
Schleife für den Schlüssel
Ausgeben: Schlüssel, Zahl für Zahl
Ausgeben: Ergebnis der Auswertung des Schlüssels (RWS, RW)


pruefen_RWS (Schloss, Schlüsselbund, Runde)
Schleife um Elemente im Schloss durchzug ehen
Wenn Schloss == Schlüssel
true
false
Zähler für RWS er höhen
Anzahl RWS im Schlüsselbund speichern


prüfen_RW(Runde, Schloss, Schlüsselbund)
Schleife um Schlüssel durchzugehen
Schleife um Schloss durchzugehen
Bedingungen für RW erfüllt?
true
false
Zähler für RW erhöhen
Stelle für weitere RW blocken
nächste Stelle im Schlüssel untersuchen
Anzahl RW im Schlüsselbund speichern

5. **Programmtest (Bild vom Ausgabebildschirm) + Testläufe**

```
Ihre Eingabe: 2 2 3 4 4

     1.Tipp: 1 2 3 4 5 - 3RWS + 1RW
     2.Tipp: 1 1 1 1 1 - 1RWS + 0RW
     3.Tipp: 2 2 2 2 2 - 0RWS + 0RW
     4.Tipp: 2 2 3 4 5 - 2RWS + 1RW
     5.Tipp: 2 2 4 3 4 - 1RWS + 2RW
     6.Tipp: 2 2 3 4 4 - 3RWS + 0RW
Ihre Eingabe: 1 $ 3 4 4
Ungueltige Eingabe bitte gebenSie nur Zahlen von 1 bis 5 ein!
Ihre Eingabe: 1 5 3 4 4

     1.Tipp: 1 2 3 4 5 - 3RWS + 1RW
     2.Tipp: 1 1 1 1 1 - 1RWS + 0RW
     3.Tipp: 2 2 2 2 2 - 0RWS + 0RW
     4.Tipp: 2 2 3 4 5 - 2RWS + 1RW
     5.Tipp: 2 2 4 3 4 - 1RWS + 2RW
     6.Tipp: 2 2 3 4 4 - 3RWS + 0RW
     7.Tipp: 1 5 3 4 4 - 5RWS + 0RW
Glueckwunsch - Sie haben die richtige Ziffernfolge erraten!

------------------
(program exited with code: 0)

Drücken Sie eine beliebige Taste . . .
```

Edit: (im Text: „Ungueltige Eingabe…“ fehlt ein Leerzeichen nach dem „geben“. Fehler wurde behoben!)

```
C:\WINDOWS\SYSTEM32\cmd.exe
Ihre Eingabe: 1 2 3 4 5

     1.Tipp: 1 2 3 4 5 - 0RWS + 3RW
Ihre Eingabe: -1 2 3 4 5
Ungueltige Eingabe bitte geben Sie nur Zahlen von 1 bis 5 ein!
Ihre Eingabe: 0 2 3 4 5
Ungueltige Eingabe bitte geben Sie nur Zahlen von 1 bis 5 ein!
Ihre Eingabe: 2 3 4 5 6
Ungueltige Eingabe bitte geben Sie nur Zahlen von 1 bis 5 ein!
Ihre Eingabe: Text
Ungueltige Eingabe bitte geben Sie nur Zahlen von 1 bis 5 ein!
Ihre Eingabe: a 2 3 4 5
Ungueltige Eingabe bitte geben Sie nur Zahlen von 1 bis 5 ein!
Ihre Eingabe: 1000000 2 3 4 5
Ungueltige Eingabe bitte geben Sie nur Zahlen von 1 bis 5 ein!
Ihre Eingabe: % & . 3 2
Ungueltige Eingabe bitte geben Sie nur Zahlen von 1 bis 5 ein!
Ihre Eingabe: 1 1 1 1 1

     1.Tipp: 1 2 3 4 5 - 0RWS + 3RW
     2.Tipp: 1 1 1 1 1 - 2RWS + 0RW
Ihre Eingabe:
```

Test auf verschiedene Falsche Eingaben: negative Zahlen, 0, zu große Zahlen, Text, Buchstaben, Sonderzeichen

```c
/*************************************************************/
/* Hausarbeit Elektrotechnik, Labor Informatik Hr. Kandziora*/
/* Matrikelnummer: 9026481                                   */
/* Master Mind                                               */
/*************************************************************/

#include <stdio.h>
#include <time.h> //Bibliotheken, für den Zufallszahlengenerator
#include <stdlib.h>


/* Funktion um einen zufälligen Code in das Array Schloss zu laden*/
void zufalls_Code_Generator(int schloss[5]);

/* Funktion zum Vergleichen von "Schloss" und "Schluessel" und pruefen
 * auf RWS + speichern der Anzahl im Schlüsselbund*/
void pruefen_RWS(int z, int schloss[5], int schluesselbund[10][7]);

/* Funktion zum vergleichen von Schloss und Schluessel, und pruefen
 * auf RW + speichern der Anzhal im Schlüsselbund*/
void pruefen_RW(int z, int schloss[5], int schluesselbund[10][7]);

/* Funktion zum Einlesen der vom Spieler eingegebenen Schluessel in den
 * richtigen Platz im Schluesselbund + prüfen auf korrekte Eingaben*/
int einlesen(int z, int schluesselbund[10][7]);

/* Funktion zur Ausgabe der bisherigen Eingaben
 * inklusive Auswertung auf RWS und RW */
void ausgeben(int z, int schluesselbund[10][7]);

/* Funktion für eine Runde des Spiels MasterMind*/
int spiel(int z, int schloss[5], int schluesselbund[10][7]);

int main(int argc, char **argv){
/* anlegen von "schloss" und "schluesselbund", im Schloss liegt der zu
 * erratendende Code. Im Schluesselbund werden Eingaben und Auswertungen
 * gespeichert.*/
    int schloss [5] = { 0 };
    int schluesselbund [10][7] = { 0 };
    int spielrunde, status;

    zufalls_Code_Generator(schloss);
/* Schleife um bis zu 10 Runden zu spielen */
    for (spielrunde = 0; spielrunde < 10; spielrunde++){

        status = spiel(spielrunde, schloss, schluesselbund);

/* Neustarten der aktuellen Runde wenn Fehlerhafte Eingabe*/
        if (status == 2){
            while( getchar() != '\n') {};//Tastaurpuffer leeren
            spielrunde --;
        }
    }
    return 0;
}

void zufalls_Code_Generator(int schloss[5]){
    srand(time(NULL));
/* speichert 5 zufaellige  Zahlen von 1-5 in "schloss"*/
    for(int s=0;s<5;s++){
        schloss[s]=rand()%5 + 1;
    }
}

```

```c
void pruefen_RWS(int z, int schloss[5], int schluesselbund[10][7]){

    int s, rws = 0;

        for(s=0; s<5; s++){
            if (schloss[s]==schluesselbund[z][s])
                rws++;
        }
/*speichert die Anzahl an RWS des aktuellen Schlüssels an der Stelle 5*/
    schluesselbund[z][5] = rws;
}

void pruefen_RW(int z, int schloss[5], int schluesselbund[10][7]){
/* genauere Erlaeuterung der pruefen RW Funktion im Word Dokument*/
    int doppel[5]={ 0 };
    int sKey, sLock, rw = 0;

/* äußere Schleife geht den Schlüssel durch*/
    for(sKey=0; sKey<5; sKey++){
/* innere Schleife geht das Schloss durch*/
        for(sLock=0; sLock<5; sLock++){
/* Folgende If Abfrage überprüft, ob ein RW vorliegt*/
            if (schluesselbund[z][sKey]==schloss[sLock] &&
                schluesselbund[z][sLock]!=schloss[sLock] &&
                schluesselbund[z][sKey]!=schloss[sKey] &&
                doppel[sLock]== 0)
                {
                    doppel[sLock]= 1;
                    rw++;
/* sobald ein RW gefunden wird, bricht die aktuelle Schleife ab und die
 * nächste variable im Schluessel wird auf RW untersucht*/
                    break;
                }
        }
    }
/*speichert die Anzahl der RW in der letzten Spalte im Schluesselbund*/
    schluesselbund[z][6]= rw;
}

int einlesen(int z, int schluesselbund[10][7]){

    int eingabe, korrekteEingabe = 0;

    printf("Ihre Eingabe: ");
    for(int s=0; s<5; s++){
/*Einlesen und ueberpruefen auf korrekte Eingaben (nur Zahlen von 1-5)*/
        korrekteEingabe = scanf("%d", &eingabe);
        if (0<eingabe && eingabe<6 && korrekteEingabe == 1){
                schluesselbund[z][s] = eingabe;
        }
        else{
            printf("Ungueltige Eingabe bitte geben "
                    "Sie nur Zahlen von 1 bis 5 ein!\n");
            return 2;
        }
    }
/* leert den stdin Puffer, ansonsten wuerde es zu Problemen kommen,
 * wenn mehr als 5 Zeichen eingegeben werden*/
    while( getchar() != '\n') {};
    printf("\n");
    return 0;
}

void ausgeben(int z, int schluesselbund[10][7]){
```

```
129
130         for(int i=0; i<z+1; i++){
131                 printf("      %d.Tipp: ", i+1);
132             for(int s=0; s<5; s++){
133                 printf("%d ", schluesselbund[i][s]);//Ausgabe des Schlüssels
134             }
135             printf(" - %dRWS + %dRW\n", schluesselbund[i][5], schluesselbund[i][6]);
136         }
137     }
138
139     int spiel(int z, int schloss[5], int schluesselbund[10][7]){
140         if (einlesen(z, schluesselbund) == 2){
141             return 2;//übergeben von '2' an Main bei Fehler
142         }
143         pruefen_RWS(z, schloss, schluesselbund);
144         pruefen_RW(z, schloss, schluesselbund);
145         ausgeben(z, schluesselbund);
146
147         if(schluesselbund[z][5]==5){ //wenn 5 RWS im Schlüssel sind
148             printf("Glueckwunsch - Sie haben die richtige "
149                    "Ziffernfolge erraten!");
150             exit(1);//Beenden des Programms
151             }
152
153     /* Beenden das Spiel wenn nicht innerhalb von 10 Runden der passende
154      * Schluessel gefunden wurde + ausgeben der Lösung*/
155         if(z == 9){
156             printf("\nSie haben es leider nicht geschafft den "
157                    "Code zu erraten\n");
158             printf("Der Loesungscode ist folgender gewesen: ");
159             for(int s=0; s<5; s++)
160             printf("%d ", schloss[s]);
161         }
162         return 0;
163     }
164
```