**Dokumentation BWINF 2021 - Aufgabe 3: „Hex-Max“**

Teilnahme-ID: 61719

Bearbeiter: Karl Jahn

Dresden, der 28. Dezember 2021

**Inhalt**

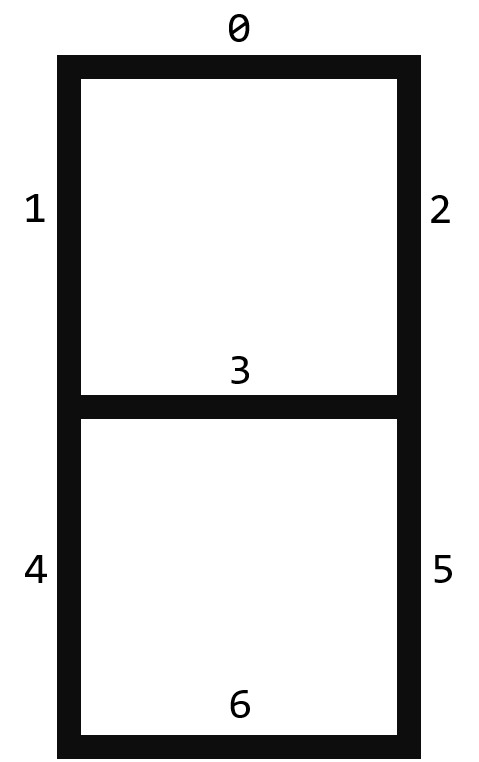
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **Beschreibung/Thema** | | **Seite** |
|  | | **Inhaltsverzeichnis** | | **1** |
| **1.** | | [**Lösungsidee**](#Lösungsidee) | | **2** |
| **2.** | | [**Umsetzung**](#Umsetzung) | | **2-4** |
|  | **2.1** | | [**Algorithmus**](#Algorithmus) | **2** |
|  | **2.2** | | [**Implementation**](#Implementation) | **3-4** |
| **3.** | | [**Quellcode**](#Quellcode) | | **5-6** |
| **4.** | | [**Beispiele**](#Beispiele) | | **7-8** |
|  |  | | [**parkplatz0.txt**](#bsp0) | **7** |
|  |  | | [**parkplatz1.txt**](#bsp1) | **7** |
|  |  | | [**parkplatz2.txt**](#bsp2) | **7** |
|  |  | | [**parkplatz3.txt**](#bsp3) | **8** |
|  |  | | [**parkplatz4.txt**](#bsp4) | **8** |
|  |  | | [**parkplatz5.txt**](#bsp5) | **8** |
| **5.** | | [**Quellen**](#Quellen) | | **9** |

**1. Lösungsidee**

Zuerst machte ich mir Gedanken drüber, wie ich die hexadezimale Zahl geeignet und effizient als Stäbchen darstellen kann; und zwar so, dass ich schnell Unterschiede zwischen zwei Zahlen herausfinden kann. In der Umsetzung werde ich genauer auf meinen Ansatz eingehen, festzuhalten ist nur, dass durch diese Art schnell herauszufinden ist, welche Stäbchen weggelegt, umgelegt, bzw. dazugelegt werden müssen.

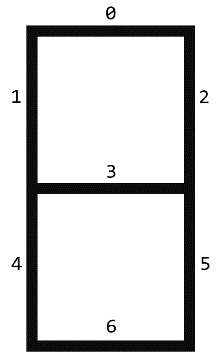
Mein erster Ansatz zur Lösung war Iterativ, nach einigen verzweifelten versuchen musste ich aber feststellen, dass ich um Rekursion nicht herumkomme. Um das höchste Ergebnis zu erzielen, sollte man logischer Weise an der vordersten Stelle anfangen und versuchen, dort das höchstmögliche zu erreichen. Sollte dort etwas fehlen oder Überschuss vorhanden sein, um eine höhere Zahl zu erzielen, so wird versucht diese von dem restlichen Teil zu holen oder dort irgendwie unterzubringen. Des Weiteren ist hierbei zu beachten, dass die maximale Anzahl der Schritte nicht zu überschreiten ist.

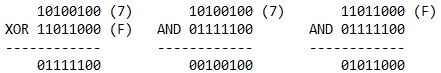
**2.** **Umsetzung**

**2.1 Datendarstellung**

Zur Darstellung einer solchen Ziffer werden maximal 7 verschiedene Stäbchen an 7 Positionen benötigt. So liegt es nahe, diese Zustände einfach Binär in einen uint8\_t (C/C++) darzustellen, dieser besitzt ein „Stäbchen“ mehr, aber das muss ja nicht beachtet werden. Sieben dieser acht Bits stellen also eine Ziffer dar.

Auf der Abbildung 1 ist zu sehen, welches Bit welche Position darstellt, das Letzte (7) ist das, was nicht beachtet wird und sollte immer null sein, da der Algorithmus die gesetzten Bits zählt. Nach dieser Darstellung sieht beispielsweise die Sieben Binär so aus: 10100100 und die Acht logischer Weise so: 11111110.

Soll nun die 7 in F umgewandelt werden, kann mittels eines XORs herausgefunden werden, welche Positionen verändert werden müssen. 7 und ein logisches AND des Ergebnisses des XOR sind die Stellen, die weggelegt werden müssen, F und das Ergebnis sind die stellen, wo Stäbe benötigt werden (auf Abb. 2 nachzuvollziehen).



Das muss geändert werden.

Das muss dazugelegt werden.

Das muss weggelegt werden.

01234567  
0b11111110

**2.2 Algorithmus**

6

**2.3 Implementation**

**4. Quellco****de**

Abb. 5: Code zur berechnung



**4. Beispieles**

Abb. 6: Beispielparkplatz 0 mit UI

Abb. 7: Beispielparkplatz 1 im Ausgabefeld

Abb. 8: Beispielparkplatz 2 im Ausgabefeld

**5.** **Quellen**

Zur Erstellung des Struktogrammes half mir dies kleine Tool: <https://dditools.inf.tu-dresden.de/ovk/Informatik/Programmierung/Grundlagen/Struktogramme.html>

Des Weiteren ist der Code und die restlichen Lösungen von mir auch auf GitHub auffindbar: <https://github.com/Tollpatsch314/BwInf/tree/Runde-2>