3.1 Elementare Ein-/Ausgabe: Hello, world (1 Punkt)

Erstellen Sie ein Programm, das "**Hello, world\n**" auf der Konsole (Standardausgabe, **stdout**) ausgibt. Nutzen Sie dafür die Ausgabefunktion `printf`.

Quellen: `hw.c`

Executable: `hw`

3.2 Elementare Ein-/Ausgabe: ASCII-Tabelle (2 Punkte)

Der American Standard Code for Information Interchange (ASCII) ist eine weitverbreite Zeichencodierung. Standardmäßig wird hierbei jeder Zahl von 0 bis 127 ein **character** zugeordnet. Die Großbuchstaben von A bis Z sind bspw. durch die Zahlen 65 bis 90 im Dezimalsystem codiert. Um einen Überblick über die enthaltenen Zeichen zu bekommen, erstellen Sie ein Programm, das folgende Tabelle für alle ASCII-Codes in verschiedenen Zahlensystemen ausgibt:

```
Oct     Dec     Hex     Char
000     0       00
...
012     10      0a

013     11      0b


...
033     27      1b
34      28      1c
...
060     48      30      0
061     49      31      1
...
101     65      41      A
102     66      42      B
...
```

Nutzen Sie für die Formatierung der Zeilen `printf` mit geeigneten Formatstring-Platzhaltern (`man 3 printf`). Achten Sie auf die verschieden formatierten Zahlen: **oktal** mit führender 0, **dezimal** linksbündig, **hexadezimal** mit 2 Stellen (a-f in Kleinbuchstaben) dargestellt. Die letzte Spalte soll die jeweiligen Characters drucken, unabhängig davon, ob diese darstellbar sind oder nicht (s. unten), eine spezielle Behandlung solcher Fälle ist nicht gefragt. Das Drucken von nicht-darstellbaren Zeichen als `char` wird hierbei (erwünschte) Nebeneffekte haben, wie im obigen Beispiel angedeutet. Die jeweiligen Spalten sollen durch Tabulatoren getrennt werden.

Quellen: `asc.c`

Executable: `asc`

Beantworten Sie sich die Frage, warum die ersten 32 Zeichen des ASCII-Codes nicht darstellbar sind. Erklären Sie Teile Ihrer Antwort mithilfe des Terminalbefehls `man ascii` **(keine Abgabe)**.

3.3 Elementare Ein-/Ausgabe: Umwandlung von Zeichen (2 Punkte)

Erstellen Sie ein Programm, das einen Namen als Zeichenkette (String) von **stdin** einliest. Nutzen Sie dafür die Funktion `read`. Wandeln Sie alle Kleinbuchstaben der eingelesenen Zeichenkette in Großbuchstaben um. Achten Sie darauf, dass Sie wirklich nur Buchstaben umwandeln: bei der Eingabe eines Zeichens außerhalb von `[a..z]` bzw. `[A..Z]` soll das Programm bis zum letzten gültigen Buchstaben ausgeführt werden.

Die resultierende Zeichenkette soll zusätzlich mittels des ROT13-Algorithmus "verschlüsselt" werden. Nutzen Sie die Funktion `write` um das Ergebnis in der Form "Hallo: [Name] -- [ROT13-Name]\n" auszugeben. Informationen zu ROT13 finden Sie im Web. Der eingegebene String soll also zuerst in Großbuchstaben umgewandelt und anschließend um 13 Stellen innerhalb der Großbuchstaben weiterrotiert werden. Achten Sie hierbei darauf, dass die Eingabe üblicherweise mit einem \n endet, welches Sie für die Ausgabe entfernen müssen. Auch hier soll die Funktion bei nicht-alphabetischen Zeichen bis zum letzten gültigen Buchstaben ausgeführt werden, wie im folgenden Beispiel zu sehen ist:

```
$ ./rotup
Ingo123!ABC                              // Eingabe
Hallo: Ingo123!ABC -- VATB               // Ausgabe
```

Quellen: `rotup.c`

Executable: `rotup`

3.4 hexdump (5 Punkte)

Schreiben Sie eine Funktion:

```
void hexdump (FILE *output, char *buffer, int length);
```

zur Ausgabe eines Puffers als "Hexadezimal-Dump". Eine solche Funktion eignet sich bestens zum Debuggen. Sie gibt einen Puffer in Form der ASCII-Codes der Zeichen im Hexadezimalformat einerseits und der entsprechenden Zeichen (sofern sie darstellbar sind) andererseits aus.

Ein Beispiel:

```
000000 : 47 72 75 6e 64 6c 61 67 65 6e 20 42 65 74 72 69   Grundlagen Betri
000010 : 65 62 73 73 79 73 74 65 6d 20 75 6e 64 20 53 79   ebssystem und Sy
000020 : 73 74 65 6d 73 6f 66 74 77 61 72 65 00            stemsoftware.
```

Ganz links steht der relative Offset zum Beginn des Puffers. Rechts werden die entsprechenden Zeichen lesbar ausgegeben, sofern ihr ASCII-Code zwischen 0x20 (dez. 32) und 0x7e (dez. 126) liegt und damit darstellbar ist. Bei Werten außerhalb dieses Bereichs wird statt des Zeichens ein Punkt (.) ausgegeben.

Das Ausgabeformat soll exakt dem folgenden Aufbau entsprechen:

- 6 Zeichen Puffer-Offset mit führenden Nullen
- 1 Leerzeichen, 1 Doppelpunkt, 1 Leerzeichen
- 16 x die Ausgabe des nächsten Bytes im Puffer formatiert in 2 Zeichen mit führender 0 im Hexadezimalformat, wobei jeweils zwei aufeinanderfolgende Ausgaben durch genau ein Leerzeichen getrennt sind
- 3 Leerzeichen
- 16 x die Ausgabe der zuvor hexadezimal dargestellten Bytes als Zeichen, sofern dieses "druckbar" ist (siehe oben)
- Newline

Beachten Sie, dass die letzte Zeile weniger als 16 Zeichen im Puffer haben kann und dass dementsprechend Leerzeichen ergänzt werden müssen, um die korrekte Darstellung zu erhalten.

Testen Sie die Funktion, indem Sie vier Character-Arrays unterschiedlicher Länge im Quellcode definieren und dann diese als Hexdump ausgeben.

Erweitern Sie Ihre Funktion so, dass der auszugebende Puffer als Kommandozeilenparameter übergeben wird. Werden mehrere übergeben, so soll das Programm jeden Puffer einzeln ausgeben, wobei jeweils zwei aufeinanderfolgende durch eine Leerzeile getrennt sind. Achten Sie darauf, dass Ihr Programm sich auch dann sinnvoll verhält, wenn kein Parameter angegeben wird.

Implementieren Sie die Funktion `hexdump ()` in einer Datei `hexdump.c` und die Funktion `main ()` in einer separaten Datei `hd.c`; denken Sie daran, in letzterer den Funktionsprototypen korrekt zu deklarieren.

Quellen: `hexdump.c hd.c`

Executable: `hd`

Beispiel:

```
$ ./hd Hello "Operating Systems" now.
000000 : 48 65 6c 6c 6f 00                                      Hello.

000000 : 4f 70 65 72 61 74 69 6e 67 20 53 79 73 74 65 6d        Operating System
000010 : 73 00                                                  s.

000000 : 6e 6f 77 2e 00                                         now..
```