Tutorübung Grundlagen: Betriebssysteme und Systemsoftware

Moritz Beckel

München, 4. November 2022

Freitag 10:15-12:00 Uhr

Raum 00.11.038

https://zulip.in.tum.de/#narrow/stream/1295-GBS-Fr-1000-A

Bei Fragen könnt ihr mich hier kontaktieren:

moritz.beckel@tum.de

Organisation

- Quizze finden auf Moodle statt und zählen zum Bonus
- Klausur ist voraussichtlich in Präsenz (Räume sind reserviert)

Zusammenfassung

- Rechensysteme (Definition, Struktur)
- Betriebssysteme

1. Definition, Aufgaben, Entwicklung, Einsatzgebiete, historische Entwicklung
2. Ressourcenklassifikation
3. Prozesse
4. Datei- und Gerätezugriff
5. Betriebssystem-Modi
6. Systemaufrufe
7. Betriebssystemarchitekturen
8. Systemprogrammierung

Zusammenfassung

Rechensysteme (Definition, Struktur)

1. Offenes System (Gliederung in Komponenten mit Schnittstellen)
2. Dynamisches System (Passive, Aktive Komponenten)
3. Technisches System (Hardware + Software)
4. Recherarchitektur (Von-Neumann)
5. Schichtenstruktur (Anwendungsprogramme, Systemprogramme, Hardware)

Zusammenfassung

Betriebssysteme

1. Definition: Steuerung, Überwachung von Anwendungsprogrammen + Schnittstelle zu Hardware
2. Aufgaben: Abstraktion, Ressourcenmanagement
3. Entwicklung: Hardwarefortschritt, neue Anwendungsbereiche (General Purpose, Special Purpose)
4. Einsatzgebiete: Server, Desktop, Mobil, Embedded
5. Betriebsarten (Stapel, Dialog, Transaktion, Echtzeit (Hard Deadlines, Soft Deadlines)
6. Historische Entwicklung

Zusammenfassung

Ressourcenklassifikation

1. Anzahl der Nutzungen
2. Parallelität
3. Unterbrechbarkeit
4. Zentrale vs. Periphere Ressourcen

Prozesse

1. Unterschied Programm und Prozess (aktiv, passiv)
2. Ressourcenverteilung (Prozessor, Speicher, IO)

Zusammenfassung

Datei- und Gerätezugriff

1. Everything is a File
2. Dateideskriptoren

Betriebssystem-Modi

1. Benutzermodus
2. Systemmodus

Zusammenfassung

Systemaufrufe (system calls)

1. Schnittstelle zum Betriebssystem
2. Wechsel vom User Mode nach Kernel Mode
3. Mittels Interrupt ausgelöst

Zusammenfassung

Betriebssystemarchitekturen

1. Monolithisches System
2. Mikrokernel
3. Hybrider Kernel

Systemprogrammierung

1. Algorithmen für Ausführung von Benutzerprogrammen
2. System-Software bietet Dienste für Anwendungssoftware

Aufgabe 1

Operatorpräzedenz

- C Operatoren binden unterschiedlich stark (bspw. * bindet stärker als +)
- Gilt auch bei Deklaration von neuen Variablen

Regel: [] (Array) und () (Funktion) werden von links nach rechts abgearbeitet und haben Vorrang vor * (Pointer/Dereferenzierung) und & (Adresse), welche von rechts nach links abgearbeitet werden.

Aufgabe 1

a) long **foo[7];

Regel: [] (Array) und () (Funktion) werden von links nach rechts abgearbeitet und haben Vorrang vor * (Pointer/Dereferenzierung) und & (Adresse), welche von rechts nach links abgearbeitet werden.

Aufgabe 1

a) long **foo[7];

- declare foo as array 7 of pointer to pointer to long

Regel: [] (Array) und () (Funktion) werden von links nach rechts abgearbeitet und haben Vorrang vor * (Pointer/Dereferenzierung) und & (Adresse), welche von rechts nach links abgearbeitet werden.

Aufgabe 1

b) unsigned long int **x();

Regel: [] (Array) und () (Funktion) werden von links nach rechts abgearbeitet und haben Vorrang vor * (Pointer/Dereferenzierung) und & (Adresse), welche von rechts nach links abgearbeitet werden.

Aufgabe 1

b) unsigned long int **x();

- x is function returning pointer to pointer to unsigned long int

Regel: [] (Array) und () (Funktion) werden von links nach rechts abgearbeitet und haben Vorrang vor * (Pointer/Dereferenzierung) und & (Adresse), welche von rechts nach links abgearbeitet werden.

Aufgabe 1

c) char *(*(**foo [][8])())[];

Regel: [] (Array) und () (Funktion) werden von links nach rechts abgearbeitet und haben Vorrang vor * (Pointer/Dereferenzierung) und & (Adresse), welche von rechts nach links abgearbeitet werden.

Aufgabe 1

c) char *(*(**foo [][8])())[];

- foo is array of array of 8 pointer to pointer to function returning pointer to array of pointer to char

Regel: [] (Array) und () (Funktion) werden von links nach rechts abgearbeitet und haben Vorrang vor * (Pointer/Dereferenzierung) und & (Adresse), welche von rechts nach links abgearbeitet werden.

Aufgabe 1

d) int (*(*foo)(void))[];

Regel: [] (Array) und () (Funktion) werden von links nach rechts abgearbeitet und haben Vorrang vor * (Pointer/Dereferenzierung) und & (Adresse), welche von rechts nach links abgearbeitet werden.

Aufgabe 1

d) int (*(*foo)(void))[];

- foo is a pointer to function with no arguments returning pointer to array of int

Regel: [] (Array) und () (Funktion) werden von links nach rechts abgearbeitet und haben Vorrang vor * (Pointer/Dereferenzierung) und & (Adresse), welche von rechts nach links abgearbeitet werden.

Aufgabe 1

- In Wahrheit sind Operatorpräzedenzregeln komplexer
- Genaue Dokumentation unter: https://en.cppreference.com/w/c/language/operator_precedence

Zusätzliche Hilfe:

- http://unixwiz.net/techtips/reading-cdecl.html
- https://cdecl.org/

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(*type*){*list*}	Compound literal(C99)	
2	++ --	Prefix increment and decrement[note 1]	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	Cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of[note 2]	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional[note 3]	Right-to-left
14[note 4]	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Aufgabe 2

Hexdump

- Direktes Lesen aus dem Speicher mit Hexwerten
- Endianness gibt die Bytereihenfolge an
- Little-Endian: low-order byte on lowest address
- Big-Endian: low-order byte on highest address

Zu klärende Fragen:

1. Länge einer Adresse? – 32 Bit
2. Little-endian oder big-endian? – little-endian
3. Welche Größe haben byte/char, int, long?
   – int = 32

```
0x0000  89 50 4e 47 0d 0a 1a 0a  ff 00 00 00 49 48 44 52
0x0010  00 00 05 00 00 00 02 82  08 06 00 00 00 8e 3b 74
0x0020  aa 00 00 00 a7 73 42 49  54 08 08 08 08 7c 08 64
0x0030  47 42 53 00 88 00 00 00  4d 00 00 0f 61 00 00 0f
0x0040  61 01 a8 3f a7 69 00 00  00 38 74 45 58 74 53 6f
0x0050  66 74 77 61 72 65 00 6d  61 74 70 6c 6f 74 6c 69
0x0060  f1 20 f5 65 72 73 69 6f  6e 33 2e 31 2e 31 2c 20
0x0070  68 74 74 70 3a 2f 73 74  72 69 6e 67 00 74 6c 69
0x0080  62 2e 6f 72 67 2f 10 66  17 19 00 00 20 00 49 44
0x0090  41 54 78 9c ec dd 77 9c  54 e5 a1 ff f1 cf 6c 85
0x00a0  5d 76 97 be 85 de 62 69  6e 67 6f 00 a4 05 93 dc
0x00b0  18 63 bb 37 a2 49 ae 29  e6 17 d4 28 22 45 16 05
0x00c0  c4 44 62 12 4d 31 d7 80  b1 5f 5b 12 51 41 20 8a
0x00d0  08 00 00 00 39 05 00 00  b2 d4 2d 2c db cf ef 8f
0x00e0  49 f6 c6 58 00 dd e5 ec  ce 7e de af d7 bc 5e 3c
0x00f0  87 39 33 df 59 75 9d f9  ce 73 9e 27 12 04 41 80
```

Aufgabe 2

a) Wie viele Hex-Zeichen umfasst eine Speicheradresse im obigen Hexdump?

```
0x0000  89 50 4e 47 0d 0a 1a 0a  ff 00 00 00 49 48 44 52
0x0010  00 00 05 00 00 00 02 82  08 06 00 00 00 8e 3b 74
0x0020  aa 00 00 00 a7 73 42 49  54 08 08 08 08 7c 08 64
0x0030  47 42 53 00 88 00 00 00  4d 00 00 0f 61 00 00 0f
0x0040  61 01 a8 3f a7 69 00 00  00 38 74 45 58 74 53 6f
0x0050  66 74 77 61 72 65 00 6d  61 74 70 6c 6f 74 6c 69
0x0060  f1 20 f5 65 72 73 69 6f  6e 33 2e 31 2e 31 2c 20
0x0070  68 74 74 70 3a 2f 73 74  72 69 6e 67 00 74 6c 69
0x0080  62 2e 6f 72 67 2f 10 66  17 19 00 00 20 00 49 44
0x0090  41 54 78 9c ec dd 77 9c  54 e5 a1 ff f1 cf 6c 85
0x00a0  5d 76 97 be 85 de 62 69  6e 67 6f 00 a4 05 93 dc
0x00b0  18 63 bb 37 a2 49 ae 29  e6 17 d4 28 22 45 16 05
0x00c0  c4 44 62 12 4d 31 d7 80  b1 5f 5b 12 51 41 20 8a
0x00d0  08 00 00 00 39 05 00 00  b2 d4 2d 2c db cf ef 8f
0x00e0  49 f6 c6 58 00 dd e5 ec  ce 7e de af d7 bc 5e 3c
0x00f0  87 39 33 df 59 75 9d f9  ce 73 9e 27 12 04 41 80
```

Aufgabe 2

a) Wie viele Hex-Zeichen umfasst eine Speicheradresse im obigen Hexdump?

- eine Speicheradresse 4 Hex Zweiergruppen breit

```
0x0000  89 50 4e 47 0d 0a 1a 0a  ff 00 00 00 49 48 44 52
0x0010  00 00 05 00 00 00 02 82  08 06 00 00 00 8e 3b 74
0x0020  aa 00 00 00 a7 73 42 49  54 08 08 08 08 7c 08 64
0x0030  47 42 53 00 88 00 00 00  4d 00 00 0f 61 00 00 0f
0x0040  61 01 a8 3f a7 69 00 00  00 38 74 45 58 74 53 6f
0x0050  66 74 77 61 72 65 00 6d  61 74 70 6c 6f 74 6c 69
0x0060  f1 20 f5 65 72 73 69 6f  6e 33 2e 31 2e 31 2c 20
0x0070  68 74 74 70 3a 2f 73 74  72 69 6e 67 00 74 6c 69
0x0080  62 2e 6f 72 67 2f 10 66  17 19 00 00 20 00 49 44
0x0090  41 54 78 9c ec dd 77 9c  54 e5 a1 ff f1 cf 6c 85
0x00a0  5d 76 97 be 85 de 62 69  6e 67 6f 00 a4 05 93 dc
0x00b0  18 63 bb 37 a2 49 ae 29  e6 17 d4 28 22 45 16 05
0x00c0  c4 44 62 12 4d 31 d7 80  b1 5f 5b 12 51 41 20 8a
0x00d0  08 00 00 00 39 05 00 00  b2 d4 2d 2c db cf ef 8f
0x00e0  49 f6 c6 58 00 dd e5 ec  ce 7e de af d7 bc 5e 3c
0x00f0  87 39 33 df 59 75 9d f9  ce 73 9e 27 12 04 41 80
```

Aufgabe 2

b) Wie lautet die Adresse, auf die der Pointer an der Adresse 0x8c zeigt?

```
0x0000  89 50 4e 47 0d 0a 1a 0a  ff 00 00 00 49 48 44 52
0x0010  00 00 05 00 00 00 02 82  08 06 00 00 00 8e 3b 74
0x0020  aa 00 00 00 a7 73 42 49  54 08 08 08 08 7c 08 64
0x0030  47 42 53 00 88 00 00 00  4d 00 00 0f 61 00 00 0f
0x0040  61 01 a8 3f a7 69 00 00  00 38 74 45 58 74 53 6f
0x0050  66 74 77 61 72 65 00 6d  61 74 70 6c 6f 74 6c 69
0x0060  f1 20 f5 65 72 73 69 6f  6e 33 2e 31 2e 31 2c 20
0x0070  68 74 74 70 3a 2f 73 74  72 69 6e 67 00 74 6c 69
0x0080  62 2e 6f 72 67 2f 10 66  17 19 00 00 20 00 49 44
0x0090  41 54 78 9c ec dd 77 9c  54 e5 a1 ff f1 cf 6c 85
0x00a0  5d 76 97 be 85 de 62 69  6e 67 6f 00 a4 05 93 dc
0x00b0  18 63 bb 37 a2 49 ae 29  e6 17 d4 28 22 45 16 05
0x00c0  c4 44 62 12 4d 31 d7 80  b1 5f 5b 12 51 41 20 8a
0x00d0  08 00 00 00 39 05 00 00  b2 d4 2d 2c db cf ef 8f
0x00e0  49 f6 c6 58 00 dd e5 ec  ce 7e de af d7 bc 5e 3c
0x00f0  87 39 33 df 59 75 9d f9  ce 73 9e 27 12 04 41 80
```

Aufgabe 2

0x8C

```
0x0000  89 50 4e 47 0d 0a 1a 0a  ff 00 00 00 49 48 44 52
0x0010  00 00 05 00 00 00 02 82  08 06 00 00 00 8e 3b 74
0x0020  aa 00 00 00 a7 73 42 49  54 08 08 08 08 7c 08 64
0x0030  47 42 53 00 88 00 00 00  4d 00 00 0f 61 00 00 0f
0x0040  61 01 a8 3f a7 69 00 00  00 38 74 45 58 74 53 6f
0x0050  66 74 77 61 72 65 00 6d  61 74 70 6c 6f 74 6c 69
0x0060  f1 20 f5 65 72 73 69 6f  6e 33 2e 31 2e 31 2c 20
0x0070  68 74 74 70 3a 2f 73 74  72 69 6e 67 00 74 6c 69
0x0080  62 2e 6f 72 67 2f 10 66  17 19 00 00 20 00 49 44
0x0090  41 54 78 9c ec dd 77 9c  54 e5 a1 ff f1 cf 6c 85
0x00a0  5d 76 97 be 85 de 62 69  6e 67 6f 00 a4 05 93 dc
0x00b0  18 63 bb 37 a2 49 ae 29  e6 17 d4 28 22 45 16 05
0x00c0  c4 44 62 12 4d 31 d7 80  b1 5f 5b 12 51 41 20 8a
0x00d0  08 00 00 00 39 05 00 00  b2 d4 2d 2c db cf ef 8f
0x00e0  49 f6 c6 58 00 dd e5 ec  ce 7e de af d7 bc 5e 3c
0x00f0  87 39 33 df 59 75 9d f9  ce 73 9e 27 12 04 41 80
```

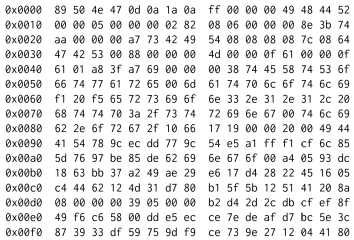
Aufgabe 2

b) Wie lautet die Adresse, auf die der Pointer an der Adresse 0x8c zeigt?

- an der Adresse 0x8c hinterlegten Bytes sind 20 00 49 44
- von der little-endian in die big-endian Darstellung 44 49 00 20

```
0x0000  89 50 4e 47 0d 0a 1a 0a  ff 00 00 00 49 48 44 52
0x0010  00 00 05 00 00 00 02 82  08 06 00 00 00 8e 3b 74
0x0020  aa 00 00 00 a7 73 42 49  54 08 08 08 08 7c 08 64
0x0030  47 42 53 00 88 00 00 00  4d 00 00 0f 61 00 00 0f
0x0040  61 01 a8 3f a7 69 00 00  00 38 74 45 58 74 53 6f
0x0050  66 74 77 61 72 65 00 6d  61 74 70 6c 6f 74 6c 69
0x0060  f1 20 f5 65 72 73 69 6f  6e 33 2e 31 2e 31 2c 20
0x0070  68 74 74 70 3a 2f 73 74  72 69 6e 67 00 74 6c 69
0x0080  62 2e 6f 72 67 2f 10 66  17 19 00 00 20 00 49 44
0x0090  41 54 78 9c ec dd 77 9c  54 e5 a1 ff f1 cf 6c 85
0x00a0  5d 76 97 be 85 de 62 69  6e 67 6f 00 a4 05 93 dc
0x00b0  18 63 bb 37 a2 49 ae 29  e6 17 d4 28 22 45 16 05
0x00c0  c4 44 62 12 4d 31 d7 80  b1 5f 5b 12 51 41 20 8a
0x00d0  08 00 00 00 39 05 00 00  b2 d4 2d 2c db cf ef 8f
0x00e0  49 f6 c6 58 00 dd e5 ec  ce 7e de af d7 bc 5e 3c
0x00f0  87 39 33 df 59 75 9d f9  ce 73 9e 27 12 04 41 80
```

Aufgabe 2



c) Bestimmen Sie die Ausgaben des folgenden Programms

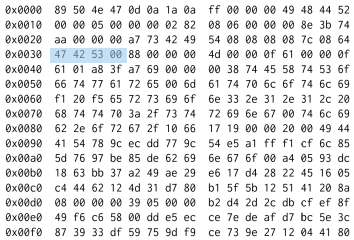
```
1     char *x = (char*) 0x30;
2     int* i = (int*) 0xd0;
3
4     printf("Some␣string:␣%s\n", x);
5     printf("Some␣other␣string:␣%s\n", x+0x46);
6
7     int a = i[1];
8     int b = *(int*)*i;
9     printf("a:␣%d,␣b:␣%d\n", a, b);
```

```
char *x = (char*) 0x30;
printf("Some string: %s\n", x);
```

```
0x0000  89 50 4e 47 0d 0a 1a 0a  ff 00 00 00 49 48 44 52
0x0010  00 00 05 00 00 00 02 82  08 06 00 00 00 8e 3b 74
0x0020  aa 00 00 00 a7 73 42 49  54 08 08 08 08 7c 08 64
0x0030  47 42 53 00 88 00 00 00  4d 00 00 0f 61 00 00 0f
0x0040  61 01 a8 3f a7 69 00 00  00 38 74 45 58 74 53 6f
0x0050  66 74 77 61 72 65 00 6d  61 74 70 6c 6f 74 6c 69
0x0060  f1 20 f5 65 72 73 69 6f  6e 33 2e 31 2e 31 2c 20
0x0070  68 74 74 70 3a 2f 73 74  72 69 6e 67 00 74 6c 69
0x0080  62 2e 6f 72 67 2f 10 66  17 19 00 00 20 00 49 44
0x0090  41 54 78 9c ec dd 77 9c  54 e5 a1 ff f1 cf 6c 85
0x00a0  5d 76 97 be 85 de 62 69  6e 67 6f 00 a4 05 93 dc
0x00b0  18 63 bb 37 a2 49 ae 29  e6 17 d4 28 22 45 16 05
0x00c0  c4 44 62 12 4d 31 d7 80  b1 5f 5b 12 51 41 20 8a
0x00d0  08 00 00 00 39 05 00 00  b2 d4 2d 2c db cf ef 8f
0x00e0  49 f6 c6 58 00 dd e5 ec  ce 7e de af d7 bc 5e 3c
0x00f0  87 39 33 df 59 75 9d f9  ce 73 9e 27 12 04 41 80
```

```
    2 3 4 5 6 7
   -----------
0:   0 @ P ` p
1: ! 1 A Q a q
2: " 2 B R b r
3: # 3 C S c s
4: $ 4 D T d t
5: % 5 E U e u
6: & 6 F V f v
7: ' 7 G W g w
8: ( 8 H X h x
9: ) 9 I Y i y
A: * : J Z j z
B: + ; K [ k {
C: , < L \ l |
D: - = M ] m }
E: . > N ^ n ~
F: / ? O _ o DEL
```

Aufgabe 2



c) Bestimmen Sie die Ausgaben des folgenden Programms

```
1     char *x = (char*) 0x30;
2     int* i = (int*) 0xd0;
3
4     printf("Some␣string:␣%s\n", x);
5     printf("Some␣other␣string:␣%s\n", x+0x46);
6
7     int a = i[1];
8     int b = *(int*)*i;
9     printf("a:␣%d,␣b:␣%d\n", a, b);
```

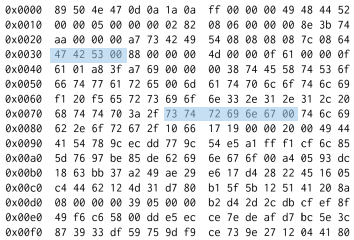
Some string: GBS

```
char *x = (char*) 0x30;
printf("Some other string: %s\n", x+0x46);
```

```
0x0000  89 50 4e 47 0d 0a 1a 0a  ff 00 00 00 49 48 44 52
0x0010  00 00 05 00 00 00 02 82  08 06 00 00 00 8e 3b 74
0x0020  aa 00 00 00 a7 73 42 49  54 08 08 08 08 7c 08 64
0x0030  47 42 53 00 88 00 00 00  4d 00 00 0f 61 00 00 0f
0x0040  61 01 a8 3f a7 69 00 00  00 38 74 45 58 74 53 6f
0x0050  66 74 77 61 72 65 00 6d  61 74 70 6c 6f 74 6c 69
0x0060  f1 20 f5 65 72 73 69 6f  6e 33 2e 31 2e 31 2c 20
0x0070  68 74 74 70 3a 2f 73 74  72 69 6e 67 00 74 6c 69
0x0080  62 2e 6f 72 67 2f 10 66  17 19 00 00 20 00 49 44
0x0090  41 54 78 9c ec dd 77 9c  54 e5 a1 ff f1 cf 6c 85
0x00a0  5d 76 97 be 85 de 62 69  6e 67 6f 00 a4 05 93 dc
0x00b0  18 63 bb 37 a2 49 ae 29  e6 17 d4 28 22 45 16 05
0x00c0  c4 44 62 12 4d 31 d7 80  b1 5f 5b 12 51 41 20 8a
0x00d0  08 00 00 00 39 05 00 00  b2 d4 2d 2c db cf ef 8f
0x00e0  49 f6 c6 58 00 dd e5 ec  ce 7e de af d7 bc 5e 3c
0x00f0  87 39 33 df 59 75 9d f9  ce 73 9e 27 12 04 41 80
```

```
   2 3 4 5 6 7
  -------------
0:   0 @ P ` p
1: ! 1 A Q a q
2: " 2 B R b r
3: # 3 C S c s
4: $ 4 D T d t
5: % 5 E U e u
6: & 6 F V f v
7: ' 7 G W g w
8: ( 8 H X h x
9: ) 9 I Y i y
A: * : J Z j z
B: + ; K [ k {
C: , < L \ l |
D: - = M ] m }
E: . > N ^ n ~
F: / ? O _ o DEL
```

Aufgabe 2



c) Bestimmen Sie die Ausgaben des folgenden Programms

```
1    char *x = (char*) 0x30;
2    int* i = (int*) 0xd0;
3
4    printf("Some␣string:␣%s\n", x);
5    printf("Some␣other␣string:␣%s\n", x+0x46);
6
7    int a = i[1];
8    int b = *(int*)*i;
9    printf("a:␣%d,␣b:␣%d\n", a, b);
```

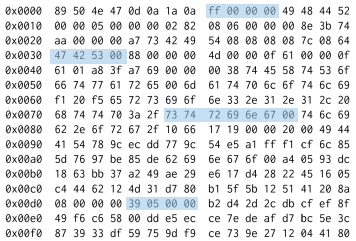
Some string: GBS

Some other string: string

```
int* i = (int*) 0xd0;
int a = i[1];
int b = *(int*)*i;
printf("a: %d, b: %d\n", a, b);
```

```
0x0000  89 50 4e 47 0d 0a 1a 0a  ff 00 00 00 49 48 44 52
0x0010  00 00 05 00 00 00 02 82  08 06 00 00 00 8e 3b 74
0x0020  aa 00 00 00 a7 73 42 49  54 08 08 08 08 7c 08 64
0x0030  47 42 53 00 88 00 00 00  4d 00 00 0f 61 00 00 0f
0x0040  61 01 a8 3f a7 69 00 00  00 38 74 45 58 74 53 6f
0x0050  66 74 77 61 72 65 00 6d  61 74 70 6c 6f 74 6c 69
0x0060  f1 20 f5 65 72 73 69 6f  6e 33 2e 31 2e 31 2c 20
0x0070  68 74 74 70 3a 2f 73 74  72 69 6e 67 00 74 6c 69
0x0080  62 2e 6f 72 67 2f 10 66  17 19 00 00 20 00 49 44
0x0090  41 54 78 9c ec dd 77 9c  54 e5 a1 ff f1 cf 6c 85
0x00a0  5d 76 97 be 85 de 62 69  6e 67 6f 00 a4 05 93 dc
0x00b0  18 63 bb 37 a2 49 ae 29  e6 17 d4 28 22 45 16 05
0x00c0  c4 44 62 12 4d 31 d7 80  b1 5f 5b 12 51 41 20 8a
0x00d0  08 00 00 00 39 05 00 00  b2 d4 2d 2c db cf ef 8f
0x00e0  49 f6 c6 58 00 dd e5 ec  ce 7e de af d7 bc 5e 3c
0x00f0  87 39 33 df 59 75 9d f9  ce 73 9e 27 12 04 41 80
```

Aufgabe 2



c) Bestimmen Sie die Ausgaben des folgenden Programms

```
1     char *x = (char*) 0x30;
2     int* i = (int*) 0xd0;
3
4     printf("Some string: %s\n", x);
5     printf("Some other string: %s\n", x+0x46);
6
7     int a = i[1];
8     int b = *(int*)*i;
9     printf("a: %d, b: %d\n", a, b);
```

Some string: GBS

Some other string: string

a: 1337, b: 255

Aufgabe 3

a) Erklären Sie wieso jeder der folgenden Programmschnipsel auf keinen Fall verwendet werden sollte.

```
char userinput[256];
gets(userinput);
```

Aufgabe 3

a) Erklären Sie wieso jeder der folgenden Programmschnipsel auf keinen Fall verwendet werden sollte.

```
char userinput[256];
gets(userinput);
```

- gets-Funktion sollte nicht verwendet werden, da sie die Länge des gelesenen Inputs nicht begrenzt (potentieller Buffer-Overflow)

Aufgabe 3

b) Erklären Sie wieso jeder der folgenden Programmschnipsel auf keinen Fall verwendet werden sollte.

```
char userinput[256] = {0};
int ret = scanf("%256s", userinput);
```

Aufgabe 3

b) Erklären Sie wieso jeder der folgenden Programmschnipsel auf keinen Fall verwendet werden sollte.

```
char userinput[256] = {0};
int ret = scanf("%256s", userinput);
```

- Falls Input Länge 256, dann passt Null-Terminator nicht mehr in das Array und überschreibt somit Daten

Aufgabe 3

c) Erklären Sie wieso jeder der folgenden Programmschnipsel auf keinen Fall verwendet werden sollte.

```
1  #define MUL(x,y) x*y
2  int y = MUL(4+1, 3+6);
```

Aufgabe 3

c) Erklären Sie wieso jeder der folgenden Programmschnipsel auf keinen Fall verwendet werden sollte.

```
1  #define MUL(x,y) x*y
2  int y = MUL(4+1, 3+6);
```

- MUL(4+1, 3+6) wird zu 4+1*3+6, korrekte Definition: #define MUL(x, y) ((x) * (y))

Aufgabe 3

d) Erklären Sie wieso jeder der folgenden Programmschnipsel auf keinen Fall verwendet werden sollte.

```
int *p1, p2;
```

Aufgabe 3

d) Erklären Sie wieso jeder der folgenden Programmschnipsel auf keinen Fall verwendet werden sollte.

```
int *p1, p2;
```

- Nur p1 ist ein Pointer, p2 hat den Typ int. p2 muss ein * vorangestellt sein

Aufgabe 3

d) Erklären Sie wieso jeder der folgenden Programmschnipsel auf keinen Fall verwendet werden sollte.

```
int *p = malloc(sizeof *p);
scanf("%d", p);
free(p);
printf("*p is %d", *p);
```

Aufgabe 3

d) Erklären Sie wieso jeder der folgenden Programmschnipsel auf keinen Fall verwendet werden sollte.

```
int *p = malloc(sizeof *p);
scanf("%d", p);
free(p);
printf("*p is %d", *p);
```

- Speicherbereich auf den p zeigt wird noch nach Freigabe verwendet (Use-After-Free)

Aufgabe 3

e) Erklären Sie wieso jeder der folgenden Programmschnipsel auf keinen Fall verwendet werden sollte.

```
int* list;
if (list == NULL) {
  list = malloc(LIST_SIZE);
}
```

Aufgabe 3

e) Erklären Sie wieso jeder der folgenden Programmschnipsel auf keinen Fall verwendet werden sollte.

```
int* list;
if (list == NULL) {
  list = malloc(LIST_SIZE);
}
```

- list-Variable wird nicht initialisiert, somit zufälliger Wert, if-statement nicht ausgeführt

Aufgabe 1 Klausur 2020

Wir befinden uns auf einer fiktiven 16-bit little-endian Architektur (die Werte werden demnach in der umgekehrten Bytefolge interpretiert). Eine Speicheradresse ist somit 16 bit breit. Ein Char sei 8 bit breit, ein Integer sei 16 bit breit, ein Long 32 bit. Auf den Speicher kann beliebig byteweise zugegriffen werden, es wird kein Alignment enforced.

```
1  int* i = (int*) 0xca;
2
3  int* a = &i[2];
4  long b = *(long*)*(int*)*i;
5  char* c = ((char*)i)+0x6;
```

a) Wie viele Hex-Zeichen umfasst eine Speicheradresse im obigen Hexdump?

b) An der Adresse 0xf4 ist ein Pointer gespeichert. Geben Sie den Pointer in big-endian als Hexadezimalwert an.

c) Welcher Wert wird der Variablen a (Zeile 3) zugewiesen? Geben Sie den Wert als Hexadezimalwert in big-endian an.

d) Welcher Wert wird der Variablen b (Zeile 4) zugewiesen? Geben Sie den Wert als Hexadezimalwert in big-endian an.

e) Geben Sie den String an, der durch die Variable c (Zeile 5) referenziert wird. (Hinweis: Nutzen Sie man 7 ascii oder Abbildung 1.1)

```
0x0020  19 ab 4d 00 80 89 22 95  5b 81 31 00 3a 00 8b 89
0x0030  58 b7 96 01 0f 00 00 ed  10 48 24 01 f8 00 01 00
0x0040  d1 63 f5 04 b0 00 c6 2f  78 c7 e6 e1 a6 fe a3 98
0x0050  01 a3 01 00 31 b0 5f 40  51 00 45 ab 60 78 b0 2e
0x0060  04 4f fa c8 44 f1 ef 1d  14 45 a5 e9 e6 76 5b 50
0x0070  23 43 87 57 00 10 9c 01  27 4d 06 a2 4b 3c 4c 28
0x0080  d8 5a 83 fc 0c c9 dd 31  d1 3c ee 8a ff 6f c8 00
0x0090  96 bf 66 d1 a3 9c 27 a2  b8 c0 d9 65 1e ec 5e ff
0x00a0  75 b3 dc d4 f3 12 78 f9  0f e2 1c 09 bc 28 00 00
0x00b0  71 3f c8 b8 3b 6b a0 87  99 00 01 a2 04 47 41 02
0x00c0  b1 00 9a a5 17 b1 01 1a  7a 6e 3c 00 63 b9 be 3c
0x00d0  7a 65 72 6f 00 82 04 a9  ca e0 93 85 2e f4 5c 3d
0x00e0  2f 31 aa 8c 2c 29 f4 ad  21 a7 6b 49 5b a0 e3 91
0x00f0  bc e7 ca 44 85 9f fc 01  21 77 8e 00 e9 32 c6 69
0x0100  76 6f 69 64 00 00 60 c9  47 60 ad 00 42 ec a7 86
0x0110  6e 65 6c 6c 00 81 8d 2d  9b 26 a5 d0 4b f9 03 51
0x0120  92 00 86 92 00 10 af b3  f6 54 f0 f5 c6 00 c5 48
0x0130  f7 58 3b 19 72 38 fb df  be ab c7 7d 8a 46 6c 5b
0x0140  eb 16 80 ab ea 7e 73 14  6f c0 f7 35 7b 65 00 15
```

```
1  int* i = (int*) 0xca;
2
3  int* a = &i[2];
4  long b = *(long*)*(int*)*i;
5  char* c = ((char*)i)+0x6;
```

a) Wie viele Hex-Zeichen umfasst eine Speicheradresse im obigen Hexdump?
   - 4

b) An der Adresse 0xf4 ist ein Pointer gespeichert. Geben Sie den Pointer in big-endian als Hexadezimalwert an.
   - 0x 85 44

c) Welcher Wert wird der Variablen a (Zeile 3) zugewiesen? Geben Sie den Wert als Hexadezimalwert in big-endian an.
   - 0x ce

d) Welcher Wert wird der Variablen b (Zeile 4) zugewiesen? Geben Sie den Wert als Hexadezimalwert in big-endian an.
   - 0x 8e 77 21

e) Geben Sie den String an, der durch die Variable c (Zeile 5) referenziert wird. (Hinweis: Nutzen Sie man 7 ascii oder Abbildung 1.1)
   - zero

```
0x0020  19 ab 4d 00 80 89 22 95  5b 81 31 00 3a 00 8b 89
0x0030  58 b7 96 01 0f 00 00 ed  10 48 24 01 f8 00 01 00
0x0040  d1 63 f5 04 b0 00 c6 2f  78 c7 e6 e1 a6 fe a3 98
0x0050  01 a3 01 00 31 b0 5f 40  51 00 45 ab 60 78 b0 2e
0x0060  04 4f fa c8 44 f1 ef 1d  14 45 a5 e9 e6 76 5b 50
0x0070  23 43 87 57 00 10 9c 01  27 4d 06 a2 4b 3c 4c 28
0x0080  d8 5a 83 fc 0c c9 dd 31  d1 3c ee 8a ff 6f c8 00
0x0090  96 bf 66 d1 a3 9c 27 a2  b8 c0 d9 65 1e ec 5e ff
0x00a0  75 b3 dc d4 f3 12 78 f9  0f e2 1c 09 bc 28 00 00
0x00b0  71 3f c8 b8 3b 6b a0 87  99 00 01 a2 04 47 41 02
0x00c0  b1 00 9a a5 17 b1 01 1a  7a 6e 3c 00 63 b9 be 3c
0x00d0  7a 65 72 6f 00 82 04 a9  ca e0 93 85 2e f4 5c 3d
0x00e0  2f 31 aa 8c 2c 29 f4 ad  21 a7 6b 49 5b a0 e3 91
0x00f0  bc e7 ca 44 85 9f fc 01  21 77 8e 00 e9 32 c6 69
0x0100  76 6f 69 64 00 00 60 c9  47 60 ad 00 42 ec a7 86
0x0110  6e 65 6c 6c 00 81 8d 2d  9b 26 a5 d0 4b f9 03 51
0x0120  92 00 86 92 00 10 af b3  f6 54 f0 f5 c6 00 c5 48
0x0130  f7 58 3b 19 72 38 fb df  be ab c7 7d 8a 46 6c 5b
0x0140  eb 16 80 ab ea 7e 73 14  6f c0 f7 35 7b 65 00 15
```

Aufgabe 2 Klausur 2017

a) Für welche Werte von i und j wird im Folgenden Equal ausgegeben, wenn i und j als int deklariert sind? Was wollte der Programmierer vermutlich schreiben / erreichen?

```
if ( i = j )
    printf("Equal\n");
```

Aufgabe 2 Klausur 2017

a) Für welche Werte von i und j wird im Folgenden Equal ausgegeben, wenn i und j als int deklariert sind? Was wollte der Programmierer vermutlich schreiben / erreichen?

```
if ( i = j )
    printf("Equal\n");
```

- Zuweisungsoperator (=) wird verwendet anstatt Vergleichsoperator (==)
- i und j haben zufällige Werte, da nicht initialisiert, printf wird sehr wahrscheinlich ausgeführt

Aufgabe 2 Klausur 2017

b) Welches Problem ergibt sich beim Aufruf von print_array?

```
void print_array(int *arr, int size)
{
    int i = 0;

    for (i=0; i<=size; ++i)
    {
        printf("arr[%d] = %d\n", i, *(arr + i));
    }
}
```

Aufgabe 2 Klausur 2017

b) Welches Problem ergibt sich beim Aufruf von print_array?

```
void print_array(int *arr, int size)
{
    int i = 0;

    for (i=0; i<=size; ++i)
    {
        printf("arr[%d] = %d\n", i, *(arr + i));
    }
}
```

- Es wird an der Stelle arr[size] zugegriffen

Aufgabe 2 Klausur 2017

c) Welches Problem ergibt sich beim Aufruf von swap?

```
void swap(int *a, int *b)
{
    int tmp = &a;
    *a = *b;
    *b = tmp;
}
```

Aufgabe 2 Klausur 2017

c) Welches Problem ergibt sich beim Aufruf von swap?

```
void swap(int *a, int *b)
{
    int tmp = &a;
    *a = *b;
    *b = tmp;
}
```

- Mit &a wird die Adresse von a nach tmp geschrieben, nicht der eigentliche Wert

Aufgabe 2 Klausur 2017

d) Erklären Sie die folgende Deklaration?

```
char (*x())[]
```

Aufgabe 2 Klausur 2017

d) Erklären Sie die folgende Deklaration?

```
char (*x())[]
```

- declare x as function returning pointer to array of char