

Assessment I

Vorname: _____

Punkte: ____ / 90, Note: ____

Name: _____

Frei lassen für Korrektur.

Klasse: 4ibb1

Hilfsmittel:

- Ein A4-Blatt handgeschriebene Zusammenfassung.
- Lösen Sie die Aufgaben jeweils direkt auf den Prüfungsblättern.
- Zusatzblätter, falls nötig, mit Ihrem Namen und Frage-Nr. auf jedem Blatt.

Nicht erlaubt:

- Unterlagen (Slides, Bücher, ...).
- Computer (Laptop, Smartphone, ...).
- Kommunikation (mit Personen, KI, ...).

Bewertung:

- Multiple Response: ☐ *Ja* oder ☐ *Nein* ankreuzen, +1/-1 Punkt pro richtige/falsche Antwort, beide nicht ankreuzen ergibt +0 Punkte; Total pro Frage gibt es nie weniger als 0 Punkte.
- Offene Fragen: Bewertet wird Korrektheit, Vollständigkeit und Kürze der Antwort.
- Programme: Bewertet wird die Idee/Skizze und Umsetzung des Programms.

Fragen zur Prüfung:

- Während der Prüfung werden vom Dozent keine Fragen zur Prüfung beantwortet.
- Ist etwas unklar, machen Sie eine Annahme und notieren Sie diese auf der Prüfung.

Erste Schritte in C

1) Welche Ausdrücke liefern die Grösse des Typs *long* in C?

Punkte: _ / 4

Zutreffendes ankreuzen:

- ☐ Ja | ☐ Nein `size_t`
- ☐ Ja | ☐ Nein `sizeof(long)`
- ☐ Ja | ☐ Nein `long.size()`
- ☐ Ja | ☐ Nein `MAX_LONG`

2) Welche Abfolge von Statements führt zu folgender Situation im Speicher?

Punkte: _ / 4



Zutreffendes ankreuzen:

- ☐ Ja | ☐ Nein `int p = 2; int q = 0; int *a[] = {&q, &p}; *(a + 1) = 3;`
- ☐ Ja | ☐ Nein `int p; int q = 2; int *a[2] = {0}; p = q; a[0] = &p; q++;`
- ☐ Ja | ☐ Nein `int q = 2; int *a[] = {0, &q}; int p = q; a[0] = &p; q++;`
- ☐ Ja | ☐ Nein `int p; int q; int *a[] = {&p, &q}; *a[0] = 2; q = p + 1;`

|

|

|

|

|

|

|

|

(Aufgabe 3 auf der nächsten Seite)

Funktionen in C

3) Gegeben den folgenden Code, welchen Wert hat k nach Aufruf von $f()$?

Punkte: _ / 4

```
int f(int *a, int b) {  
    return (*a + 1) * b;  
}  
  
int main() {  
    int i[] = {3, 5};  
    int j = 2;  
    int k = f(i, j);  
}
```

Schrittweise Begründung und Resultat hier eintragen:

4) Gegeben den folgenden Code, welche Aufrufe von $eval()$ sind erlaubt?

Punkte: _ / 4

```
int dec(int i) { return i - 1; }  
int add(int a, int b) { return a + b; }  
int eval(int a, int b, int (*op)(int, int)) { return op(a, b); }
```

Zutreffendes ankreuzen:

- ☐ Ja | ☐ Nein `eval(dec(3), 3, add);`
- ☐ Ja | ☐ Nein `eval(add(3, 3), 3, dec);`
- ☐ Ja | ☐ Nein `eval(add(3, 3), dec, add);`
- ☐ Ja | ☐ Nein `eval(eval(3, 3, add), 3, add);`

5) Schreiben Sie ein Programm *pangram*, das prüft, ob ein per Command Line übergebener Satz ein Pangramm ist, d.h. *jeder* Buchstabe des Alphabets (a-z) kommt mindestens einmal vor. Als Antwort soll *yes* oder *no* ausgegeben werden, wie hier im Beispiel. Punkte: _ / 14

```
$ ./pangram pack my box with five dozen liquor jugs  
yes  
$ ./pangram was it a car or a cat i saw  
no
```

Verwenden Sie die folgenden Calls (soweit sinnvoll), ohne *#includes* und Fehlerbehandlung:

```
int printf(const char *format, ...); // format string %s, char %c, int %d  
size_t strlen(const char *s); // calculate the length of a string
```

Idee (kurz) und C Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:

File In-/Output

6) Schreiben Sie ein Programm *reverse*, das mittels *lseek()* ein per Command Line gegebenes, beliebig grosses File, Byte-weise umkehrt, in ein neues File, wie hier im Beispiel. P.kte: _ / 12

```
$ echo -n "hello" > my.txt
$ ./reverse my.txt your.txt
$ cat your.txt
olleh$
```

Verwenden Sie die folgenden Calls (soweit sinnvoll), ohne *#includes* und Fehlerbehandlung:

```
off_t lseek(int fd, off_t offset, int from); // position read/write file
offset; from = SEEK_SET, SEEK_CUR or SEEK_END; returns new offset from 0.
```

```
int open(const char *pathname, int flags, .../* mode_t mode */); // Opens
the file specified by pathname. Or creates it if O_CREAT is used. Returns
the file descriptor. Flags include O_CREAT, O_TRUNC, O_RDONLY, O_WRONLY.
Modes, which are used together with O_CREAT include S_IRUSR and S_IWUSR.
```

```
ssize_t read(int fd, void *buf, size_t n); // attempts to read up to n
bytes from file descriptor fd into buf. Returns number of bytes read ≤ n.
```

```
ssize_t write(int fd, const void *buf, size_t n); // writes up to n bytes
from buf to the file referred to by fd. Returns nr. of bytes written ≤ n.
```

Idee (kurz) und C Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Frage-Nr.:

7) Welche der folgenden Aussagen zu File In-/Output sind korrekt?

Punkte: _ / 4

Zutreffendes ankreuzen:

- ☐ Ja | ☐ Nein Ein File Deskriptor identifiziert ein File über Funktionsaufrufe hinweg.
- ☐ Ja | ☐ Nein Der Return-Wert von *read(fd, buf, n)* ist genau *n*, oder -1 bei Fehlern.
- ☐ Ja | ☐ Nein *STDOUT_FILENO* ist ein File Deskriptor, auf den man schreiben kann.
- ☐ Ja | ☐ Nein Die Einträge in der Open Files Tabelle zeigen auf File Deskriptoren.

Prozesse und Signale

8) Welche der folgenden Aussagen zum Heap sind korrekt?

Punkte: _ / 4

Zutreffendes ankreuzen:

- ☐ Ja | ☐ Nein Der Heap hat den virtuellen Speicher des Prozesses für sich allein.
- ☐ Ja | ☐ Nein Auf dem Heap allozierte Variablen überdauern Funktionsaufrufe.
- ☐ Ja | ☐ Nein Heap-Speicher ist begrenzt, *malloc()* kann auch fehlschlagen.
- ☐ Ja | ☐ Nein Am oberen Rand des Heaps beginnt jeweils direkt der Stack.

|
|
|
|
|
|
|
|
|
|

(Aufgabe 9 auf der nächsten Seite)

9) Schreiben Sie ein Programm *forksig*, das *SIGUSR1* vom Parent zum Child sendet, dann im Child die PID des Parents, und zuletzt im Parent die PID des Childs ausgibt. Punkte: _ / 14

```
$ ./forksig
I got SIGUSR1 from parent 75019.
Child 75020 got SIGUSR1 from me.
```

Verwenden Sie die folgenden Calls (soweit sinnvoll), ohne *#includes* und Fehlerbehandlung:

```
void exit(int status); // cause normal process termination

pid_t fork(void); // create a child process, returns 0 in child process

pid_t getpid(void); pid_t getppid(void); // get the (parent) process ID

int kill(pid_t pid, int sig); // send signal to process; r. 0 on success

int pause(void); // sleep until a signal causes invocation of a handler

int printf(const char *format, ...); // format string %s, char %c, int %d

typedef void (*sighandler_t)(int); e.g. SIGINT = 2, ^C; SIGTSTP = 20, ^Z
sighandler_t signal(int sig, sighandler_t h); // set h to handle a signal

pid_t wait(int *wstatus); // wait for child process to terminate; returns
the process ID of the terminated child or -1 if no child left to wait for
```

Idee (kurz) und C Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Frage-Nr.:

Prozess Lebenszyklus

10) Schreiben Sie ein Programm *runseq*, welches per Command Line übergebene Programme nacheinander ausführt, bzw. den Status *-1* zurückgibt, wenn eine Ausführung fehlschlägt, und am Schluss die Anzahl erfolgreiche Ausführungen anzeigt, wie unten im Beispiel. P.kte: / 14

```
$ ./runseq hello hxllo hello
Hello, World!
execve: No such file or directory
Hello, World!
runseq: 2 of 3 runs successful
```

Verwenden Sie die folgenden Calls, ohne *#includes*, mit Fehlerbehandlung soweit nötig:

```
int execve(const char *pathname, char *const argv[], char *const envp[]);
// executes the program referred to by pathname; argv, envp can be NULL

void exit(int status); // cause normal process termination

pid_t fork(void); // create a child process, returns 0 in child process

int printf(const char *format, ...); // format string %s, char %c, int %d

pid_t wait(int *wstatus); // wait for child process to terminate; returns
the process ID of the terminated child or -1 if no child left to wait for
```

Idee (kurz) und C Source Code hier, oder auf Zusatzblatt mit Ihrem Namen & Frage-Nr.:

Threads und Synchronisation

11) Gegeben das folgende Programm, was sind die zwei wesentlichen Probleme im sichtbaren

Teil des Codes, und wie kann man diese mit minimalen Änderungen beheben? Punkte: _ / 8

```

01 int n;
02 volatile int g = 0;
03
04 pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
05
06 int f(void) {
07     pthread_mutex_lock(&m);
08     int res = g + 1; // read g
09     pthread_mutex_unlock(&m);
10     return res;
11 }
12
13 void *start(void *arg) {
14     pthread_mutex_lock(&m);
15     for (int i = 0; i < n; i++) {
16         g = f(); // write g
17     }
18     pthread_mutex_unlock(&m);
19 }
20
21 int main(int argc, char *argv[]) {
22     ... // set n from argv, create 2 threads running start, join both
23 }

```

Probleme und jeweilige Behebung hier eintragen; Annahme: #includes sind vorhanden.

	Problem:	Behebung:
1		
2		

12) Was sind zwei wesentliche Vorteile von Threads gegenüber Prozessen?

Punkte: _ / 4

Vorteile hier eintragen, jeweils einen ganzen Satz ausformulieren:

1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1
16	1
17	1
18	1
19	1
20	1
21	1
22	1
23	1
24	1
25	1
26	1
27	1
28	1
29	1
30	1
31	1
32	1
33	1
34	1
35	1
36	1
37	1
38	1
39	1
40	1
41	1
42	1
43	1
44	1
45	1
46	1
47	1
48	1
49	1
50	1
51	1
52	1
53	1
54	1
55	1
56	1
57	1
58	1
59	1
60	1
61	1
62	1
63	1
64	1
65	1
66	1
67	1
68	1
69	1
70	1
71	1
72	1
73	1
74	1
75	1
76	1
77	1
78	1
79	1
80	1
81	1
82	1
83	1
84	1
85	1
86	1
87	1
88	1
89	1
90	1
91	1
92	1
93	1
94	1
95	1
96	1
97	1
98	1
99	1
100	1

(Zusatzblatt auf der nächsten Seite)

Zusatzblatt zu Aufgabe Nr. ____ von (Name) _____