

Assessment I

Vorname: _____

Punkte: ____ / 90, Note: ____

Name: _____

Frei lassen für Korrektur.

Klasse: 3ib

Hilfsmittel:

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

Nicht erlaubt:

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

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 der folgenden Aussagen sind korrekt?

Punkte: ___ / 4

Zutreffendes ankreuzen

- ☐ Ja | ☐ Nein Ein *int* Wert ist im Speicher jedes Computers genau 4 Byte gross.
- ☐ Ja | ☐ Nein Der *sizeof* Operator liefert für *int* und *float* immer dieselbe Grösse.
- ☐ Ja | ☐ Nein Variablen des Typs *int* kann man Werte des Typs *float* zuweisen.
- ☐ Ja | ☐ Nein Der Typ *int* belegt doppelt so viele Bytes wie der Typ *unsigned int*.

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

Punkte: ___ / 4



Zutreffendes ankreuzen

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

|

|

|

|

|

|

(Aufgabe 3 auf der nächsten Seite)

Funktionen in C

3) Schreiben Sie ein Programm *max*, welches das längste der *n* übergebenen Argumente, oder wie ganz unten, das erste von mehreren maximal langen Argumenten, ausgibt. Punkte: _ / 12

```
$ ./max
$ ./max short looong
looong
$ ./max short words use them
short
```

Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:

```
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 Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:

4) Gegeben den folgenden Code, implementieren Sie die Funktion *append()*, welche ein neues Item hinten an die unsortierte Liste *list* hängt.

Punkte: _ / 10

```
#include ... // ignore

struct item {
    char name[32];
    struct item* next;
};

struct item *create_item(char *name) {
    struct item *result = malloc(sizeof(struct item));
    strcpy(result->name, name);
    result->next = NULL;
    return result;
}

int equals(char *a, char *b) {
    return strcmp(a, b) == 0;
}

void append(struct item **list, struct item *i); // TODO: implement

int main() {
    struct item *list = NULL;
    append(&list, create_item("Dog"));
    append(&list, create_item("Cat"));
    append(&list, create_item("Bat"));
    assert(equals(list->next->next->name, "Bat"));
}
```

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

File In-/Output

5) Schreiben Sie ein Programm *tee*, welches Bytes von *STDIN_FILENO* liest und diese sowohl auf *STDOUT_FILENO* schreibt, als auch in eine Datei *dest* kopiert. Zudem soll das Programm anzeigen, wie es benutzt werden muss, falls es ohne Argumente aufgerufen wird. P.kte: _ / 12

```
$ ./tee
usage: ./tee dest
$ echo "hello" | ./tee log.txt
hello
$ cat log.txt
hello
```

Verwenden Sie dazu die folgenden System Calls, Fehlerbehandlung können Sie weglassen:

```
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_APPEND, 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 Source Code hier, oder auf Zusatzblatt mit Ihrem Namen & Fragen-Nr.:

Prozesse und Signale

6) Welche Vorteile hat virtueller Speicher?

Punkte: ____ / 6

Zutreffendes ankreuzen:

- | | |
|---|--|
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Daten müssen dadurch nicht auf die Disk geschrieben werden. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Dadurch muss man nicht den ganze Adressraum ins RAM laden. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Man muss das physische Speicherlayout nicht genau kennen. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Prozess bzw. laufendes Programm hat die CPU für sich allein. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Programm-Text kann zwischen Prozessen geshared werden. |
| <input type="checkbox"/> Ja <input type="checkbox"/> Nein | Der ganze virtuelle Adressraum ist gültig, verhindert SIGSEGV. |

7) Schreiben Sie ein Programm *sigseq*, welches als Argument eine Folge von Signal IDs < 32 nimmt und nur dann terminiert, wenn es genau diese Signalfolge empfangen hat. P.kte: _ / 12

```
$ ./sigseq 2 2 20 2  
^C^C^Z^C$
```

Verwenden Sie dazu die folgenden System Calls, Fehlerbehandlung können Sie weglassen:

```
int atoi(const char *nptr); // convert a string to an integer  
  
int pause(void); // Pause causes the calling process to sleep until a  
signal terminates the process or causes invocation of a handler function.  
  
typedef void (*sighandler_t)(int); e.g. SIGINT = 2, ^C; SIGTSTP = 20, ^Z  
sighandler_t signal(int sig, sighandler_t handler); // set SIG_IGN,  
SIG_DFL, or a programmer-defined function to handle the signal sig.
```

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

Prozess Lebenszyklus

8) Welche dieser Aussagen sind korrekt?

Punkte: __ / 6

Zutreffendes ankreuzen:

- ☐ Ja | ☐ Nein Ein Child Prozess kann zum Parent Prozess werden.
- ☐ Ja | ☐ Nein Rückgabewert von *fork()* ist immer eine Prozess ID.
- ☐ Ja | ☐ Nein Der Speicher des Parents wird beim *fork()*-en kopiert.
- ☐ Ja | ☐ Nein Ein Parent lebt immer länger als seine Child Prozesse.
- ☐ Ja | ☐ Nein Ein Parent Prozess kann zum Zombie Prozess werden.
- ☐ Ja | ☐ Nein Der *init* Prozess kann einen Child Prozess "adoptieren".

9) Schreiben Sie ein Programm *pos*, das durch Ausgabe der PID und der aktuellen Offset

Position im File beweist, dass ein vom Parent geöffnetes und benutztes File nach einem *fork()* auch im Child Prozess zugänglich ist, und der Offset an derselben Position steht. P.kte: __ / 12

```
$ ./pos my.txt
777: 1
778: 1
```

Verwenden Sie dazu die folgenden System Calls, Fehlerbehandlung können Sie weglassen:

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

```
pid_t getpid(void); // returns the process ID of the calling process
```

```
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_APPEND, O_CREAT, O_TRUNC, O_RDONLY, O_WRONLY.  
Modes, which are used together with O_CREAT include S_IRUSR and S_IWUSR.
```

```
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.
```

Fortsetzung auf der nächsten Seite.

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

(Aufgabe 10 auf der nächsten Seite)

Threads

10) Gegeben den folgenden Code, welcher ein Postschalter mit Warteraum-Tickets simuliert, implementieren Sie die Funktion *queue()* für Kunden die ein Ticket nehmen und warten, und *serve()* für die Person am Schalter, die ein Ticket nach dem anderen bedient. Punkte: _ / 12

Annahme: Implementierung ohne Synchronisation, und mit "busy-wait", ist hier gut genug.

```
#include ... // ignore

volatile int n = 0; // ticket no
volatile int s = 0; // served no

void *queue(void *arg); // TODO: implement taking a ticket, waiting
void *serve(void *arg); // TODO: implement serving tickets, forever

int main() {
    pthread_t server;
    pthread_create(&server, NULL, serve, NULL);
    pthread_detach(server);
    while (1) {
        pthread_t client;
        pthread_create(&client, NULL, queue, NULL);
        pthread_detach(client);
    }
}
```

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

Zusatzblatt zu Aufgabe Nr. ____ von (Name) _____