

# 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 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 dieser Ausdrücke kompilieren fehlerfrei und ohne Warnung in C? Punkte: \_\_\_ / 6

*Zutreffendes ankreuzen; Annahmen: Keine #includes; Compiliert mit gcc -std=c99*

- ☐ Ja | ☐ Nein      `bool b = !0;`
- ☐ Ja | ☐ Nein      `float f = 420 * 0.1;`
- ☐ Ja | ☐ Nein      `double d = sizeof(int);`
- ☐ Ja | ☐ Nein      `string s = "Hello, World!";`
- ☐ Ja | ☐ Nein      `char msg[] = { 'h', 'o', 'i' };`
- ☐ Ja | ☐ Nein      `byte data[] = { 0x68, 0x6f, 0x69, 0x00 };`

2) Nennen Sie einen Vor- und einen möglichen Nachteil von *unsigned* Typen? Punkte: \_\_\_ / 4

Vorteil:

Nachteil:

3) Welche der folgenden Aussagen sind immer korrekt?

Punkte: \_\_\_ / 4

*Zutreffendes ankreuzen*

- ☐ Ja | ☐ Nein      Ein *int* Wert ist im Speicher mindestens 4 Byte gross.
- ☐ Ja | ☐ Nein      Der *sizeof* Operator liefert den Wertebereich eines Typs.
- ☐ Ja | ☐ Nein      Ein *double* Wert ist weniger Bytes gross als ein *long* Wert.
- ☐ Ja | ☐ Nein      Der Typ *double* nutzt doppelt so viele Bytes wie der Typ *float*.

4) Welche Abfolge von Statements führt zu folgender Situation im Speicher? Punkte: \_\_\_ / 4



Zutreffendes ankreuzen

- ☐ Ja | ☐ Nein      `int i = 3; int *p = &i; int *q = &p;`
- ☐ Ja | ☐ Nein      `int i = 3; int *p = &i; int *q = p;`
- ☐ Ja | ☐ Nein      `int i = 2; int *p = &i; *p += 1; int *q = &i;`
- ☐ Ja | ☐ Nein      `int i = 2; int *p = &i; p += 1; int *q = p;`

## Funktionen in C

5) Implementieren Sie eine Funktion `append()`, die einen String `src` hinten an den String `dest` anhängt und `dest` zurück gibt, für beliebige Strings, ohne Hilfsfunktionen. Punkte: \_\_\_ / 12

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

// Idee:

6) Gegeben den folgenden Code, beschreiben Sie (in der Tabelle unten) vier wesentliche

Unterschiede der Funktionen `create_point1()` und `create_point2()`.

Punkte: \_\_\_ / 8

```
#include ... // ignore

struct point {
    int x;
    int y;
};

struct point create_point1(int x, int y) {
    struct point result = { x, y };
    return result;
}

struct point *create_point2(int x, int y) {
    struct point *result = malloc(sizeof(struct point));
    result->x = x;
    result->y = y;
    return result;
}

int main() {
    struct point p = create_point1(0, 0);
    struct point *q = create_point2(0, 0);
}
```

Unterschiede hier eintragen, jeweils beide Seiten des Unterschieds ausformulieren:

<code>create_point1()</code>	<code>create_point2()</code>

7) Entwerfen Sie eine Funktion `gps_get_pos()` welche zwei float Werte *lat* und *lng* liefert, sowie 0 im Erfolgs- und -1 im Fehlerfall – ohne Structs zu verwenden. Schreiben Sie nur die Deklaration hin, keine Implementierung. Zeigen Sie, wie man die Funktion in `main()` aufruft, inkl. Fehlerbehandlung, und geben Sie die Position (*lat*, *lng*) mit `printf()` aus. Punkte: \_\_\_ / 12

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

```
void exit(int status); // cause process termination, with status 0 or -1  
  
int printf(const char *format, ...); // format string %s, int %d, float %f
```

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

```
// Idee:
```

8) Gegeben den folgenden Code, implementieren Sie die Funktion *add()*, welche einen neuen Node vorne in die unsortierte Liste *list* hängt und diese zurück gibt. Sowie die Funktion *get()*, welche den Node mit Key *key* in der Liste *list* findet und diesen zurückgibt. Punkte: \_\_\_\_ / 12

```
#include ... // ignore

struct node {
    int key;
    char value[32];
    struct node* next;
};

struct node *list = NULL;

struct node *add(struct node *list, int key, char *value);
struct node *get(struct node *list, int key);

int main() {
    list = add(list, 3000, "Bern");
    list = add(list, 4000, "Basel");
    list = add(list, 8000, "Zurich");
    struct node *n = get(list, 4000);
    printf("%d: %s\n", n->key, n->value);
}
```

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

```
void *malloc(size_t size); // Allocates size bytes, returns pointer to
the allocated memory.
```

```
char *strcpy(char *dest, char *src); // Copies src to dest, incl. '\0'.
```

*Idee (kurz) und Source Code hier und auf Folgeseite, oder Blatt mit Namen und Fragen-Nr.:*

```
// Idee:
```

```
// Fortsetzung auf Folgeseite
```

(8) Fortsetzung:

## File In-/Output

9) Schreiben Sie ein Programm *halve*, welches eine beliebige Datei *source* halbiert, und die zweite Hälfte der Datei in eine neue Datei *dest* verschiebt, wie hier im Beispiel. Punkte:    / 12

```
$ echo "Hello, World!" > source
$ ./halve source dest
$ cat source
Hello,
$ cat dest
World!
```

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

```
int ftruncate(int fd, off_t length); // Truncate a file to length bytes.
```

```
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); // Opens the file specified by
pathname. Flags include O_APPEND, O_CREAT, O_RDONLY, O_WRONLY, O_RDWR.
```

```
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.
```

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

// Idee :



## Prozesse und Signale

10) Gegeben den folgenden Code, erklären Sie wieso printf() den Wert 23 ausgibt. P.kte: \_\_ / 8

```
01: #include <stdio.h>
02:
03: void f() {
04:     int i = 23;
05: }
06:
07: void g() {
08:     int j;
09:     printf("%d\n", j);
10: }
11:
12: int main() {
13:     f();
14:     g();
15: }
```

*Erklärung hier eintragen, Schritt für Schritt ausformulieren:*

11) Schreiben Sie ein Programm *raise* welches sich selbst das Signal SIGUSR1 genau dreimal sendet, dazwischen jeweils dessen Empfang abwartet, und dann terminiert. Punkte: \_\_\_ / 8

```
int pause(void); // Pause causes the calling process to sleep until a
signal terminates the process or causes invocation of a handler function.
```

```
int raise(int sig); // Send the signal sig to the calling process.
```

```
typedef void (*sighandler_t)(int);
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.:*

```
// Idee:
```

*Zusatzblatt zu Aufgabe Nr. \_\_\_\_ von (Name) \_\_\_\_\_*