Einführung in die Programmierung
WS 2016/2017
Blatt 9

Alex Hirsch, Simon Hangl, Sebastian Stabinger

2017–01–17

- Abgabe bis spätestens Montag 21:59 über OLAT (`https://lms.uibk.ac.at/olat/dmz/`).
- Bereiten Sie jede Aufgabe so vor, dass Sie Ihre Lösung im Proseminar präsentieren können!
- Benennen Sie Ihre Abgabe nach folgendem Schema: *Gruppennummer*-*Nachname*-`blatt`*Übungsblattnummer*`.tar.gz` Wenn Sie also Max Mustermann heißen und Gruppe 1 besuchen, heißt die Datei von Übung 9: `1-mustermann-blatt9.tar.gz`
- Compilieren Sie alle Programme mit den Optionen `-Wall -Werror -std=c99`

Feedback

Nutzen Sie die angebotenen Möglichkeiten, uns Feedback zu geben (eMail, Tutorium, Proseminar). Hier können Sie uns auf Probleme, notwendige Stoffwiederholungen, Unklarheiten, aber auch positive Dinge, die beibehalten werden sollten, hinweisen.

Testen und Dokumentation

Stellen Sie sicher, dass alle Lösungen fehlerfrei kompilieren. Testen Sie Ihre Lösungen ausführlich (z.B. auf falsche Eingaben, falsche Berechnungen, Sonderfälle) und dokumentieren Sie sie. Dies hilft Ihnen bei der Präsentation und uns beim Nachvollziehen Ihrer Entscheidungen. Lesen Sie die Aufgaben *vollständig* durch.

Aufgabe 1 (4 Punkte)

In der folgenden Aufgabe soll eine Funktion für arithmetische Operationen (Addition, Multiplikation) auf jedem n-ten Element eines Arrays programmiert werden.

- Erstellen Sie dazu ein `enum` names `operations` mit den möglichen Werten `ADDITION` und `MULTIPLICATION`.
- Definieren Sie mittels `#define` eine Konstante `MAX_ELEM_COUNT` mit dem Wert 10.
- Legen Sie in der `main` Funktion ein Array mit dieser Länge an.
- Implementieren Sie weiters die Funktion `arithmetic` mit folgenden Paremetern:
  - `arr`: Ein Array von ganzen Zahlen
  - `length`: Länge des Arrays
  - `n`: Parameter n aus der obigen Beschreibung
  - `operation`: Enum, das definiert, ob addiert oder multipliziert werden soll.

Füllen Sie das Array programmatisch (ohne `scanf`) mit Zahlen und testen Sie Ihr Programm **ausführlich** mit verschiedenen Werten für n.

Hinweis: *Mit einem Array {3, 5, 8, 1, 10, 20} und* $n = 2$ *soll bei einer Addition zum Beispiel das Ergebnis* 21 *(3 + 8 + 10) zurückgegeben werden. Bei Multiplikation ist das Ergebnis* 240 *(3 * 8 * 10)*

Hinweis: *Abgabe: 1-mustermann-a1.c*

Aufgabe 2 (4 Punkte)

Erstellen Sie ein `struct` welches ein Array der Größe `100` (`CAPACITY`), sowie einen Member `length` beinhaltet.

Folgende Funktionen sollen von Ihnen implementiert werden (verwenden Sie den unten angegebenen Quellcode als Vorlage und ändern sie die Signaturen der Funktionen nicht):

`list_get` Gibt den Wert an der Stelle `index` zurück, falls `index` außerhalb der Liste liegt, soll `0` zurückgegeben werden.

`list_push_back` Hängt ein neues Element mit dem Wert `value` an das Ende der Liste an (sofern noch die Liste noch nicht voll ist). Gibt `true` / `false` zurück je nachdem ob das neue Element in die Liste eingefügt werden konnte oder nicht.

`list_pop_back` Entfernt das letzte Element einer Liste und gibt dieses zurück. Falls die Liste leer ist soll `0` zurückgegeben werden.

`list_print` Gibt den Inhalt der Liste auf dem Bildschirm aus. Eine Liste welche die Zahlen 1, 2 und 3 enthält soll z.B. folgendermaßen ausgegeben werden: `[1,2,3]`

Testen Sie Ihre Implementierung ausführlich. Berücksichtigen Sie im speziellen Randfälle wie eine leere, oder eine volle Liste.

```c
#define CAPACITY 100

struct List {
    /* TODO */
};

int list_get(const struct List* list, int index) { /* TODO */ }

bool list_push_back(struct List* list, int value) {
    /* TODO */
}

int list_pop_back(struct List* list) { /* TODO */ }

void list_print(const struct List* list) { /* TODO */ }

int main(void) {
    /* TODO */
    return EXIT_SUCCESS;
}
```

Hinweis: *Abgabe: 1-mustermann-a2.c*

Aufgabe 3 (4 Punkte)

Gegeben ist folgendes `struct`:

```
struct Person {
    char firstname[64];
    char lastname[64];
    int age;
};
```

Implementieren Sie *Vergleichs-Operatoren* welche 2 Personen miteinander vergleichen. Jede Funktion soll eine Eigenschaft der 2 Personen (`firstname`, `lastname` oder `age`) vergleichen.

```
int Person_cmp_firstname(const void* x, const void* y) { /* TODO */ }

int Person_cmp_lastname(const void* x, const void* y ) { /* TODO */ }

int Person_cmp_age(const void* x, const void* y) { /* TODO */ }
```

Lesen Sie sich die man-page der Funktion `qsort` durch. Die Signaturen der Vergleichs-Operatoren wurde so gewählt, dass sie mit dieser Funktion kompatibel sind.

Erstellen Sie anschließend ein Program, welches 3 Personen vom User einliest (jeweils `firstname`, `lastname` und `age`). Danach soll der User auswählen können, nach welcher Eigenschaft er die 3 Personen sortiert haben will. Das Program sortiert dann entsprechend (mittels `qsort`) und gibt die 3 Personen (sortiert) aus.

Hinweis: *Abgabe: 1-mustermann-a3.c*