Einführung in die Programmierung
WS 2015/2016
Blatt 9

Simon Hangl, Sebastian Stabinger, Benedikt Hupfauf, Johannes Kessler

2015–12–16

- Abgabe bis spätestens Dienstag 21:59:59 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)

Schreiben Sie eine Funktion `void pow_int(int* base, int exponent)`, die Potenz $p = \text{base}^{\text{exponent}}$ berechnet. Das Ergebnis soll über den Pointer `base` und nicht mittels `return` zurückgegeben werden. Entwickeln Sie dasselbe Verhalten für die Funktionen `void pow_float(float* base, int exponent)` und `void pow_double(double* base, int exponent)`.

Mit der vorherigen Vorgehensweise ist für jeden neuen Datentyp eine neue Funktion notwendig. Eine alternative Methode ist, für diesen Zweck `void*` zu verwenden. Schreiben Sie also eine zusätzliche Funktion mit der Signatur `void pow_void(void* base, int exponent, int type)`, die alle 3 Fälle behandelt. Warum wird hier ein zusätzlicher Parameter `int type` benötigt?

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

Aufgabe 2 (6 Punkte)

Deklarieren Sie ein Array `queue` von ganzen Zahlen mit 100 Elementen. Implementieren Sie mit diesem Array eine Queue[1]. Dazu müssen Sie folgende Funktionen implementieren:

1. `int enqueue(int arr[], int* lastElementIdx, int length, int element)`

   Fügt den Wert `element` in das Array `arr` mit der Länge `length` (in diesem Fall 100) hinter der Stelle `lastElementIdx` ein. Die Funktion gibt `true` zurück, wenn das Element eingefügt werden konnte und `false` falls für das Element kein Platz mehr war. Warum muss `lastElementIdx` als Pointer übergeben werden?

2. `int dequeue(int arr[], int* lastElementIdx)`

   Entfernt das erste Element in der Queue und gibt es als Returnwert zurück. Falls kein Element in der Queue ist, soll `0` zurückgegeben werden.

3. `void printQueue(const int arr[], int lastElementIdx)`

   Gibt die gesamte Queue am Bildschirm aus, ohne Sie zu verändern.

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

[1] https://de.wikipedia.org/wiki/Warteschlange_(Datenstruktur)