Einführung in die Programmierung, WS 2016/2017
Blatt 6

Simon Hangl, Sebastian Stabinger, Alex Hirsch

2016–11–29

- 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 6: `1-mustermann-blatt6.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 dieser Aufabe geht es um das Konzept der Rekursion. Bei einer Rekursion ruft sich eine Funktion selbst auf. Lesen Sie sich online entsprechende Artikel durch, bis Sie das Konzept verstanden haben. [1]

Implementieren Sie anschließend eine Additionsfunktion `int sum(int a, int b)` für $a, b \in \mathbb{N}$, welche rekursiv die zwei Zahlen `a` und `b` addiert. Dazu dürfen Sie nur den unären `++` Operator und nicht `+` verwenden. Rekursiv ist Addition wie folgt definiert:

$$\text{sum}(a, b) \begin{cases} \text{sum}(a, b-1) + 1 & b > 0 \\ a & \text{else} \end{cases} \tag{1}$$

Hinweis: *int sum(int a, int b) { return a + b; } ist nicht die Lösung!*

Überlegen Sie sich eine auch rekursive Varianten für Multiplikation und Fakultät [2] und implementieren Sie entsprechend die Funktionen `mult` und `factorial`.

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

Aufgabe 2 (6 Punkte)

Gegeben ist folgender Ausschnitt des Speichers (siehe Ende dieser Aufgabe). Links steht die Adresse, rechts der Inhalt des Bytes an dieser Adresse. Gehen Sie davon aus, dass der `int`-Typ, sowie Zeiger **8 Bit** lang sind. Der `long int` hat in unserem Fall **16 Bit**.

- Die Variable `int a` liegt auf Adresse 13. Was gibt also `printf("%d", a)` aus?
- Gegeben sei die Variable `int d`. Der Befehl `printf("%d", d)` gibt `113` aus. Was wird also `printf("%p", &d)` ausgeben?
- Der Zeiger `int* p` liegt auf Adresse 22. Auf welche Adresse verweist dieser? Was wird bei `printf("%p", p)` und bei `printf("%d", *p)` ausgegeben?
- Gegeben sei der Pointer `int* pc`. Der Befehl `printf("%d", *pc)` gibt `136` auf dem Bildschirm aus. Auf welche Adresse verweist der Zeiger? An welcher Adresse liegt der Zeiger?

[1] `https://de.wikipedia.org/wiki/Rekursive_Programmierung`
[2] `https://de.wikipedia.org/wiki/Fakultät_(Mathematik)`

- Der Pointer `long int* lp` liegt auf Adresse 9. Welche Länge in Bytes hat dieser Pointer. Welche Länge hat der Speicherbereich, auf den er verweist. Was wird der Befehl `printf("%ld", *lp)` ausgeben.

- Wir erzeugen eine neue Variable `int c = 64` und der Compiler legt diese auf Adresse 28. Anschließend erzeugen wir einen Zeiger `int* pc = &c`. Spielen Sie Compiler und überlegen Sie sich eine gültige Adresse für den Zeiger und füllen Sie den festgelegten Speicher mit dem korrekten Wert. Gehen Sie dabei davon aus, dass alle Speicherbereiche mit dem Wert 0, die bisher nicht verwendet wurden, frei sind.

  Hinweis: *Für ein besseres Verständnis notieren sie links von jeder Speicheradresse den Typ des hier gespeicherten Objekts falls er nach den obigen Angaben bekannt ist. Zeichnen Sie für jeden Zeiger einen Pfeil zu der Speicheradresse auf die er verweist.*

  Hinweis: *Auf realen Systemen schreibt der C-Standard vor, dass Integers einen Wertebereich abdecken müssen, der über 8 Bit hinausgeht. Die Annahme, dass Integers 8 Bit lang sind wird hier wegen der besseren Lesbarkeit getroffen.*

Adresse	Inhalt (Binär)	
...	...	
08	00011110	
09	00011001	
10	00000000	
11	00000000	
12	00000100	
13	00001101	
14	00000001	
15	00000000	
16	00000000	
17	00000000	
18	01111000	
19	00000000	
20	00100110	
21	00000000	
22	00010000	
23	00000000	
24	00000000	
25	00100000	
26	00000000	
27	00010010	
28	00000000	
29	01110001	`113`
30	10001000	
...	...	

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

Aufgabe 3 (4 Punkte)

Schreiben Sie eine Funktion mit gegebener Signatur

```
bool safe_division(const int* x, const int* y, int* result)
```

welche die beiden argumente x und y dereferenziert und durcheinander dividiert. Ist einer (oder mehrere) der 3 pointer ein *null-pointer*, oder der Divisor gleich 0, so soll `false` zurückgegeben werden. Andernfalls soll das berechnete Ergebnis an die Stelle gespeichert werden, wo `result` hinzeigt und `true` zurückgegeben werden.

Erklären Sie weiters welchen Einfluss `const` in diesem Fall hat. Und warum es hier angebracht ist.

Was ist eine *Pure Function*?

Würden Sie `safe_division` als *Pure Function* bezeichnen? Argumentieren Sie.

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