

Angewandte Informatik

Übungsaufgaben – 7. Januar 2016

Aufgabe 1: Landau-Symbole

Von welcher Ordnung (Landau-Symbol) sind die folgenden Problemstellungen?
Begründen Sie jeweils Ihre Antwort.

- (a) Iterative Berechnung einer Fakultät (siehe Übungsaufgabe 1 vom 19. 11. 2015)
(1 Punkt)
- (b) Rekursive Berechnung einer Fakultät
(1 Punkt)
- (c) Türme von Hanoi (nötige Anzahl von Verschiebungen in Abhängigkeit von der Anzahl der Scheiben)
(1 Punkt)
- (d) `push()` auf einen Stack
(1 Punkt)
- (e) Einfügen eines Elements irgendwo in den Stack („mitten in den Stapel schieben“)
(1 Punkt)
- (f) `push()` auf einen FIFO
(1 Punkt)

Aufgabe 2: Länge von Strings

(Diese Übungsaufgabe ist eine Ergänzung zu Aufgabe 1 der Probeklausur vom 22. 10. 2015.)

Strings werden in der Programmiersprache C durch Zeiger auf `char`-Variable realisiert.

Beispiel: `char *hello_world = "Hello,_world!\n"`

Die Systembibliothek stellt eine Funktion `strlen()` zur Ermittlung der Länge von Strings zur Verfügung (`#include <string.h>`).

Wir betrachten nun die folgenden Funktionen:

```
int fun_1 (char *s)
{
    int x = 0;
    for (int i = 0; i < strlen (s); i++)
        x += s[i];
    return x;
}
```

```
int fun_2 (char *s)
{
    int i = 0, x = 0;
    int len = strlen (s);
    while (i < len)
        x += s[i++];
    return x;
}
```

- (g) Von welcher Ordnung (Landau-Symbol) sind die beiden Funktionen hinsichtlich der Anzahl ihrer Zugriffe auf die Zeichen im String – und warum? (2 Punkte)
- (h) Von welcher Ordnung (Landau-Symbol) ist Ihre selbstgeschriebene, effizientere Funktion und warum? (1 Punkt)

Aufgabe 3: Dynamisches Bit-Array

Schreiben Sie die folgenden Funktionen zur Verwaltung eines dynamischen Bit-Arrays:

- **void bit_array_init (int n)**
Das Array initialisieren, so daß man n Bits darin speichern kann.
Die Array-Größe n ist keine Konstante, sondern erst im laufenden Programm bekannt.
Die Bits sollen auf den Anfangswert 0 initialisiert werden.
- **void bit_array_set (int i, int value)**
Das Bit mit dem Index i auf den Wert $value$ setzen.
Der Index i darf von 0 bis $n - 1$ gehen; der Wert $value$ darf 1 oder 0 sein.
- **void bit_array_flip (int i)**
Das Bit mit dem Index i auf den entgegengesetzten Wert setzen,
also auf 1, wenn er vorher 0 ist, bzw. auf 0, wenn er vorher 1 ist.
Der Index i darf von 0 bis $n - 1$ gehen.
- **int bit_array_get (int i)**
Den Wert des Bit mit dem Index i zurückliefern.
Der Index i darf von 0 bis $n - 1$ gehen.
- **void bit_array_resize (int new_n)**
Die Größe des Arrays auf new_n Bits ändern.
Dabei soll der Inhalt des Arrays, soweit er in die neue Größe paßt, erhalten bleiben.
Neu hinzukommende Bits sollen auf 0 initialisiert werden.
- **void bit_array_done (void)**
Den vom Array belegten Speicherplatz wieder freigeben.

Bei Bedarf dürfen Sie den Funktionen zusätzliche Parameter mitgeben, beispielsweise um mehrere Arrays parallel verwalten zu können. (In der objektorientierten Programmierung wäre dies der implizite Parameter `this`, der auf die Objekt-Struktur zeigt.)

Die Bits sollen möglichst effizient gespeichert werden, z. B. jeweils 8 Bits in einer `uint8_t`-Variablen.

Die Funktionen sollen möglichst robust sein, d. h. das Programm darf auch bei unsinnigen Parameterwerten nicht abstürzen, sondern soll eine Fehlermeldung ausgeben.

(14 Punkte)