Algorithmen und Datenstrukturen

07. Übungsblatt

Zeitraum: 28. November – 02. Dezember 2016

Übung 1 (AGS 3.2.39)

(a) Schreiben Sie ein C-Programm, welches den Nutzer zur Eingabe einer Zahl $n \geq 0$ auffordert. Ist $n \geq 2$, so sollen die Primfaktoren von n ausgegeben werden. Für $n = 0$ bzw. $n = 1$ soll keine Ausgabe erfolgen.
Beispiel: Ist $n = 20$, dann soll `"2 2 5 "` ausgegeben werden.
Nehmen Sie für die Teilaufgaben (b) und (c) folgenden Datentyp für Binärbäume an.

```
typedef struct node *tree;
struct node {
  int value;
  tree left;
  tree right;
};
```

(b) Ein Binärbaum `t` ist *balanciert,* falls an jedem Knoten `n` von `t` die Höhen des linken und des rechten Teilbaums von `n` maximal um 1 voneinander abweichen.
Schreiben Sie eine Funktion `int isBalanced(tree t)`, welche `1` zurückgibt, falls `t` balanciert ist, und sonst `0`. Falls Sie eigene Hilfsfunktionen verwenden, geben Sie diese vollständig an!

(c) Implementieren Sie eine Funktion `tree makeBalanced(int n)`, welche einen balancierten Binärbaum (siehe (b)) mit genau `n` Knoten anlegt. Die Werte an den Knoten des Baums können Sie dabei beliebig wählen. Nutzen Sie die Funktion `malloc` zur Allokation von Speicher.

Übung 2 (AGS 3.2.35)

Gegeben sei die folgende Typdefinition für binäre Bäume:

```
typedef struct node *tree;
typedef struct node {
    tree left, right;
} node;
```

(a) Der *Rang* eines Baumes t vom Typ `tree` ist die Länge des Pfades von der Wurzel von t bis zum am weitesten rechts stehenden Blatt von t. Schreiben Sie eine Funktion `int rank(tree t)`, welche den Rang des übergebenen Baums `t` berechnet! Für `t == NULL` soll `rank(t) == 0` sein.

(b) Ein Baum t heißt *Linksbaum*, falls für jeden Knoten v von t der Rang (vgl. **(a)**) des linken Teilbaums von v größer oder gleich dem Rang des rechten Teilbaums von v ist. Schreiben Sie eine Funktion `int isLeftist(tree t)`, welche für einen Baum `t` als Eingabe den Wert 1 ausgibt, falls `t` ein Linksbaum ist, ansonsten den Wert 0.

(c) Die Summe aller natürlichen Zahlen kleiner oder gleich 16, die Vielfache von 3 oder von 5 sind, ist $3+5+6+9+10+12+15 = 60$. Implementieren Sie eine Funktion `int s(int n)`, welche für den Parameter `n` die Summe aller natürlichen Zahlen kleiner oder gleich `n` ausgibt, welche Vielfache von 3 oder von 5 sind! Tipp: `n` ist genau dann durch `m` teilbar, wenn `n % m == 0` gilt.

Zusatzaufgabe 1 (AGS 3.2.38 ⋆)
Für die folgenden Teilaufgaben gelten folgende `#includes`, und Typdefinitionen für Listen und Binärbäume:

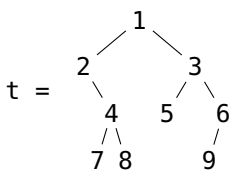
```
typedef struct elem *list;
typedef struct elem {
  int value;
  list next;
} elem;

typedef struct node *tree;
typedef struct node {
  int value;
  tree left, right;
} node;
```

(a) Schreiben Sie ein Programm, welches den Nutzer wiederholt zur Eingabe einer ganzen Zahl auffordert, bis er eine `0` eingibt. Danach soll das Programm ausgeben, wie viele gerade und wie viele ungerade Zahlen der Nutzer eingegeben hat. Die abschließende `0` soll dabei nicht mitgezählt werden.

(b) Schreiben Sie die Funktion `void histogram(list l, int h[], int length)`, welche für alle `i` mit $0 \leq$ `i` $<$ `length` den Array-Eintrag `h[i]` durch die Anzahl der Vorkommen von `i` in `l` ersetzt. *Die Liste soll dabei nur einmal durchlaufen werden.* Gehen Sie davon aus, dass `h` genau `length` Einträge hat, treffen Sie jedoch keine Annahmen über die Einträge von `h` oder `l`.

(c) Ein Pfad eines Baumes ist die Liste aller Knotenbeschriftungen auf dem Weg von der Wurzel des Baumes zu einem Knoten. (Daraus folgt, dass der leere Baum keinen Pfad enthält und jeder Pfad mindestens ein Element enthält, nämlich die Beschriftung des Wurzelknotens.)
Implementieren Sie die Funktion `int isPath(list l, tree t)`, welche `1` zurück gibt, wenn der übergebene Baum einen als Liste übergebenen Pfad beinhaltet, und ansonsten `0` zurück gibt.
Beispiel: (Listen werden vereinfacht als Sequenz der Elemente in eckigen Klammern dargestellt.)




```
isPath([], t) = 0
isPath([1, 2], t) = 1
isPath([1, 2, 7], t) = 0
isPath([1, 2, 4, 7], t) = 1
```

Zusatzaufgabe 2 (AGS 4.24)
Gegeben sei folgendes C-Programm:

```
 1  #include <stdio.h>
 2
 3  void f(int* v, int w) {
 4    /* label1 */
 5    if (w > 0) {
 6      *v = *v + 1;
 7      f(v, w - 1); /* $1 */
 8      /* label2 */
 9    }
10  }
11
12  void g(int* x, int y) {
13    while (y > 0) {
14      /* label3 */
15      f(x, y); /* $2 */
16      --y;
17    }
18    /* label4 */
19  }
20
21  int main() {
22    int a, m;
23    a = 2;
24    scanf("%i", &m);
25    /* label5 */
26    g(&a, m); /* $3 */
27    /* label6 */
28    printf("%d", a);
29    return 0;
30  }
```

(a) Erstellen Sie eine Tabelle aller Objekte und tragen Sie den Gültigkeitsbereich jedes Objektes ein. Nutzen Sie dazu die Zeilennummern.

(b) Setzen Sie das folgende Speicherbelegungsprotokoll fort. Dokumentieren Sie die aktuelle Situation beim Passieren der Marken (*label1* bis *label6*). Geben Sie jeweils den Rücksprungmarkenkeller und die *sichtbaren* Variablen mit ihrer Wertebelegung an. Die Inhalte von Speicherzellen nicht sichtbarer Variablen müssen Sie nur bei Änderungen eintragen. Die bereits festgelegten Rücksprungmarken sind *$1* bis *$3*.

Label	Rücksprungmarken	1	2	3	4	5	6	7	8	9	10
label5	–	a 2	m 2								