OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG
INFORMATIK UND MATHEMATIK

Übungsaufgaben Blatt 11 - Rekursion und Refactoring

PG 1 - Einführung in die Programmierung mit C - Prof. Dr. Ruben Jubeh

---

In den folgenden Aufgaben sollen Sie pro Aufgabe jeweils eine Funktion implementieren.

Achtung:
Für jede der folgenden Rekursionsaufgaben finden Sie in Grips ein Testprogramm. Diese Testprogramme enthalten die zu implementierende Funktion, rufen die Funktionen auf und prüfen, ob Sie die Funktionen korrekt implementiert haben. Nach jedem Kompilieren und Testaufruf können Sie sehen, ob Ihre Lösung die Tests bestanden hat. Erst wenn alle Testfälle erfolgreich bestanden wurden, wurde die Aufgabe korrekt gelöst.

Laden Sie sich für jede der Aufgaben die entsprechende Datei aus Grips (z.B. `bunnyEarsTest.c`) und öffnen diese in Clion. Programmieren und Testen Sie Ihre Lösung in den Testprogrammen.

Aufgabe 1 : Bunny Ears mit Rekursion

Gegeben ist eine Menge von Hasen. Hierbei hat jeder Hase zwei große flauschige Ohren. Berechnen Sie die Anzahl der Ohren für alle Hasen. Lösen Sie die Aufgabe rekursiv und verwenden Sie in dieser Lösung keine Schleifen und keine Multiplikation.
Ihr Programm sollte wie folgt funktionieren:
bunnyEars(0) → 0
bunnyEars(1) → 2
bunnyEars(2) → 4

```c
int bunnyEars(int bunnies) {
  /* Your code goes here, return number of ears */
  return 0;
}
```

Aufgabe 2 : Bunny Ears 2 mit Rekursion

In dieser Aufgabe stehen die Hasen in einer nummerierten Schlange (1, 2, 3, ...). Die Hasen an einer ungeraden Position (1, 3, ...) haben alle zwei Ohren. Die Hasen an einer geraden Position (2,4, ...) haben alle drei Ohren. Berechnen sie rekursiv die Anzahl der Ohren in der Hasenschlange. Verwenden Sie dabei keine Schleifen oder Multiplikation.

Ihr Programm sollte wie folgt funktionieren:
bunnyEars2(0) → 0
bunnyEars2(1) → 2

bunnyEars2(2) → 5

```
int bunnyEars2(int bunnies) {
  /* Your code goes here, return number of ears */
  return 0;
}
```

Aufgabe 3 : **Count7 mit Rekursion**

Gegeben ist eine nichtnegative Zahl. Geben Sie zurück, wie oft die Ziffer 7 in der Zahl vorkommt. Beispielsweise liefert die 717 das Ergebnis 2. Verwenden Sie in Ihrer Lösung keine Schleifen.

Hinweis: Modulodivision (%) durch 10 liefert die Ziffer ganz rechts in Ihrer Zahl (126 % 10 ist 6). Eine Division (/) durch 10 entfernt die Ziffer ganz rechts in Ihrer Zahl (126 / 10 ist 12). Ihr Programm sollte wie folgt funktionieren:

count7(717) → 2
count7(7) → 1
count7(123) → 0

```
int count7(int n) {
  /* Your code goes here, return number of ears */
  return 0;
}
```

Aufgabe 4 : **Count8 mit Rekursion**

Gegeben ist eine nichtnegative Zahl. Geben Sie zurück, wie oft die Ziffer 8 in der Zahl vorkommt. Es gilt die folgende Ausnahme: Eine 8, deren unmittelbarer linker Nachbar wieder eine 8 ist zählt doppelt. Beispielsweise ergibt 8818 das Ergebnis 4.

Hinweis: Moduludivision (%) durch 10 liefert die Ziffer ganz rechts in Ihrer Zahl (126 % 10 ist 6). Eine Division (/) durch 10 entfernt die Ziffer ganz rechts in Ihrer Zahl (126 / 10 ist 12). Ihr Programm sollte wie folgt funktionieren:

count8(8) → 1
count8(818) → 2
count8(8818) → 4

```
int count8(int n) {
  /* Your code goes here, return number of ears */
  return 0;
}
```

Aufgabe 5 : **Array6 mit Rekursion**

Gegeben ist ein Array aus Ganzzahlen. Gehen Sie davon aus, dass Sie immer nur den Teil des Arrays ab einem Startindex betrachten müssen. Somit kann ein rekursiver Aufruf `index+1` übergeben um die nächste Position des Arrays zu überprüfen. Der erste Aufruf

startet beim Index `0`. Achtung: Wenn Sie mehrere *6er* finden, sollen Sie trotzdem nur eine 1 zurückgeben, nicht die Anzahl der *6er*.
Ihr Programm sollte wie folgt funktionieren:
array6(1, 6, 4, 0) → 1
array6(1, 4, 0) → 0
array6(1, 9, 4, 6, 6, 0) → 1
array6(6, 0) → 1

```
int array6(int nums[], int startIndex, int length) {
  /* Your code goes here, return number of ears */
  return 0;
}
```

Aufgabe 6 : **Bubble Sort mit Strings und dynamischer Speicherallokation**
Schreiben ein Programm, das Strings vom Nutzer einlesen und sortieren kann. Die Strings sollen aufsteigend sortiert werden (von a nach z), benutzen Sie dazu die Funktion `strcmp()`. Bei der Eingabe *Alle meine Entchen* ist die sortierte Ausgabe dann `Alle Entchen meine`. Der Benutzer darf zu Beginn wählen wieviele Strings er sortieren will. Jeder String ist jeweils bis zu 50 Zeichen lang. Die sortierten Strings werden am Ende ausgegeben.

Hinweise zur Bearbeitung:

- Überlegen Sie welche Syntax Sie benötigen, um ein Array aus Strings zu definieren, für das Sie den Speicher dynamisch allokieren können.
- Sie finden eine Dokumentation der Funktion `strcmp()` hier: `http://en.cppreference.com/w/c/string/byte/strcmp`
- Damit Sie `strcmp()` verwenden können, müssen Sie die Bibliothek `string.h` in Ihr Programm einbinden.
- Vergessen Sich nicht den dynamisch allokierten Speicher wieder freizugeben.

Die Ausgabe sollte wie folgt aussehen:

Aufgabe 7 : **Befreundete Zahlen**

Zwei verschiedene natürliche Zahlen, von denen wechselseitig jeweils eine Zahl gleich der Summe der echten Teiler der anderen Zahl ist, bilden ein Paar befreundeter Zahlen (Quelle Wikipedia: `https://de.wikipedia.org/wiki/Befreundete_Zahlen`).
Die kleinsten befreundeten Zahlen sind 220 und 284, da die Summe der echten Teiler von 220 284 ergibt und die Summe der echten Teiler von 284 220 ergibt.

Schreiben Sie ein Programm, das die die befreundeten Zahlen bis 1.000.000 ermittelt und ausgibt. Eine mögliche Ausgabe könnte wie folgt sein: