Woche 08: Hausaufgabe 04

Allgemeines

Die zu erstellenden Funktionen sollen in der Datei `task.c` definiert werden. Innerhalb der Datei existieren bereits die Funktionsköpfe und Rümpfe (leer - sollen von euch gefüllt werden). Hierbei ist zu beachten:

- Die Funktionsköpfe dürfen **NICHT** verändert werden. Dies gilt auch für die Argumente.
- Die zugehörige Headerdatei `task.h` ist mit den entsprechenden Funktionsdeklarationen zu füllen. Zusatzfunktionen dürfen nicht enthalten sein.
- Innerhalb der `task.c` ist es verboten, eine `main`-Funktion zu erstellen. Testet die Funktionen innerhalb der gegebenen Datei `test.c`. Innerhalb der Datei `test.c` könnt ihr machen was ihr möchtet.
- Es dürfen **KEINE** globalen Variablen verwendet werden.
- Abzugeben sind nur die Datei `task.h` und `task.c`. Wir rufen die Funktionen in unserer eigenen `main`-Funktion auf. Es werden auch nur diese Dateien bewertet.
- Solange nicht **EXPLIZIT** gefordert, sind die Nutzung von Userausgaben (printf etc.) und Usereingaben (scanf etc.) innerhalb der zu erstellenden Funktionen untersagt.
- Der Code wird von uns mit `gcc -Wall -Wextra <sourcedateien>` kompiliert. Dieser zeigt euch Warnungen an, d.h. der Compiler wird sich beschweren, jedoch eine ausführbare Datei, sofern keine Fehler vorhanden sind, erzeugen.
- Abgegebene .c Dateien, welche nicht kompiliert werden können, werden mit 0 Punkten bewertet.
- Alle Funktionsköpfe , auch wenn sie nicht gelöst werden, `müssen` in der Header Datei enthalten sein, d.h. alle Funktion müssen in einer möglichen `main` aufrufbar sein.
- Eine Funktion, welche einen Rückgabewert besitzt (`void` zählt nicht dazu), **muss** ein `return` Statement mit einem passenden Wert besitzen.

Aufgabe (1) Aufsteigende Zahlen — 1 Punkt

Schreiben Sie eine Funktion, welche bei einem Aufruf einen hochzählenden Rückgabewert besitzt. Beim ersten Aufruf soll die Funktion den Rückgabewert Null zurückliefern. Der zweite Aufruf liefert den Rückgabewert Eins, der dritte den Rückgabewert Zwei usw..

Nutzen Sie hierfür den Funktionskopf der `task.c`:
`int increasingSemester(void)`

Beispielaufruf:

```
...
int val;
val = increasingSemester();
// val hat den Wert 0
val = increasingSemester();
// val hat den Wert 1
val = increasingSemester();
// val hat den Wert 2
...
```

Aufgabe (2) 2D-Arrays vergleichen — 1 Punkt

In dieser Aufgabe soll eine Funktion geschrieben werden, welche zwei 2D-Arrays `a` und `b`, `a,b` $\in \mathbb{R}^{10\times10}$ miteinander vergleicht. Stimmen zwei Elemente überein, so soll in das 2D-Array `res` eine Eins am entsprechenden Index eingetragen werden. Stimmen die Elemente an der entsprechenden Stelle nicht überein, so soll eine Null eingetragen werden.

Nutzen Sie hierfür den Funktionskopf der `task.c`:
`void compareArray(double a[10][10], double b[10][10], res[10][10])`

Beispielaufruf:

```
...
double array1[10][10] = {{0}};
double array2[10][10] = {{0}};
int isequal[10][10];
compareArray(array1,array2,isequal);
// Alle Elemente in isequal sind 1
```

Aufgabe (3) Arrayindex prüfen — 1 Punkt

Die Funktion soll prüfen, ob ein valider Index für ein Array entsprechender Länge genutzt wird. Hierzu wird der Funktion die Arraylänge und der Index übergeben, auf welchen zugegriffen wird. Ist der Zugriff außerhalb des Arrays, so soll eine 1 von der Funktion zurückgegeben werden. Ist der Zugriff valide, so soll eine 0 zurückgegeben werden.

Nutzen Sie hierfür den Funktionskopf der `task.c`:
`int checkBounds(int arraylength, int indexvalue)`

Beispielaufruf:

```
1 ...
2 int i;
3 int a[5];
4 int len = sizeof(a)/sizeof(int);
5 // Beispiel
6 i = checkBounds(len,5);
7 // i ist 1
8 i = checkBounds(len,4);
9 // i ist 0
```

Aufgabe (4) Existiert eine Datei? — 1 Punkt

Die Funktion soll prüfen, ob eine Datei existiert. Existiert die gesuchte Datei, so soll die Funktion den Wert 0 zurückgeben, ansonsten den Wert 1. Beachten Sie, dass sie nach der Arbeit mit einer Datei, diese entsprechend zu behandeln ist.

Nutzen Sie hierfür den Funktionskopf der `task.c`:
`int fexist(char *filename)`

Beispielaufruf:

```
1 // Datei kekse.txt existiert nicht
2 int i = fexist("kekse.txt");
3 // i = 1
4
5 // Datei ergebnisse.csv existiert
6 i = fexist("ergebnisse.csv");
7 // i = 0
```

Hinweis: *Im Terminal → man 3 fopen*

Aufgabe (5) Binomialkoeffizienten (Zusatz) — 1 Punkt

In dieser Aufgabe sollen Binomialkoeffizienten in ein Array `coeff` mit der entsprechenden Schranke `n` gespeichert werden. Die Formel zur Berechnung der Koeffizienten sieht wie folgt aus:

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

Hinweis: $\binom{n}{k}$ *gelesen wie: n über k*

Für diese Aufgabe kann eine Hilfsfunktion `faculty` erstellt werden. Hierfür ist in der `task.c` ein entsprechender Bereich im Abschnitt `TASK 5` auskommentiert. Beachten sie die Hinweise. Eine mögliche Implementierung finden sie im Skript unter 1.3.3.7 oder in den Vorlesungsvideos.

Die Koeffizienten sollen für $0 \leq k \leq n$ berechnet und in `coeff` gespeichert werden. Wobei `n` im Bereich $\texttt{n} \in [0, ..., 10]$ liegen soll. Eine Prüfung hierfür ist jedoch nicht notwendig.

Nutzen Sie hierfür den Funktionskopf der `task.c`:
`void binomialcoefficients(int coeff[], int n)`

Beispielaufruf

```
1         ...
2         // == N = 0 ====================
3         int n = 0
4         int coeffN0[1];
5
6         binomialcoefficients(coeffN0,n);
7         // coeffN0[0] = 1
8
9         // == N = 1 ====================
10        n = 1
11        int coeffN1[2] = {0};
12
13        binomialcoefficients(coeffN1,n);
14        // coeffN1[0] = 1
15        // coeffN1[1] = 1
16
```

```
17          // == N = 5 ====================
18          n = 5
19          int coeffN5[6] = {0};
20
21          binomialcoefficients(coeffN2,n);
22          // coeffN5[0] = 1
23          // coeffN5[1] = 5
24          // coeffN5[2] = 10
25          // coeffN5[3] = 10
26          // coeffN5[4] = 5
27          // coeffN5[5] = 1
```

Hinweis: *Pascal'sches Dreieck*