Diese Aufgabenliste soll Ihnen Anregungen geben, was Sie programmieren können. Die Komplexität der Aufgaben steigt. Bearbeiten Sie die Aufgaben entsprechend ihrer Fähigkeiten, die Sie sich im Rahmen der Vorlesung nach und nach aneignen.

Funktionen

1. Schreiben Sie ein Programm, dass Rauten auf den Bildschirm schreibt
   n=2

   o*o
   ***
   o*o

   n=3

   oo*oo
   o***o
   *****
   o***o
   oo*oo

2. Schreiben Sie ein Programm, das den größten gemeinsamen Teiler (ggT) zweier Integer-Zahlen u und v berechnet. Verwenden Sie dabei die Tatsache, daß, wenn u größer als v ist, der ggT durch den ggT von v und u−v gegeben ist.

3. Schreiben Sie ein Programm, das die Euler'sche Zahl e bis zu einer vorgegebenen Genauigkeit berechnet. Verwenden Sie dabei die Tatsache, dass gilt

   $$e = \lim_{n \to \infty} a_n$$

   mit

   $$a_n = \sum_{i=0}^{n} \frac{1}{n!}$$

   Dabei ist n! = n * (n − 1)! = 1 * 2 * ... * n die Fakultät von n.

   Die Genauigkeit (z.B. 0,01) können Sie bestimmen, indem Sie den aktuellen Wert – den vorherigen Wert berechnen:

   0! = 1

0! + 1! = 2	Genauigkeit 2-1	= 1	> 0,01	
0! + 1! + 2! = 2,5 ^	Genauigkeit 2,5-2	= 0,5	> 0,01	
0! + … + 3! = 2,666	Genauigkeit 2,66-2,5	= 0,166	> 0,01	
0! + … + 4! = 2,7083	Genauigkeit 2,70-2,66	= 0,041	> 0,01	
0! + … + 5! = 2,7167	Genauigkeit 2,71-2,70	= 0,008	< 0,01	-> Abbruch

Arrays

1. Schreiben Sie ein Programm, in dem 2 int Arrays ia und ib der Länge 5 mit unterschiedlichen Zahlen definiert werden.
   Deklarieren sie noch ein weiteres Array der Länge 5 ic.

Übergeben Sie diese Arrays und die Länge der Arrays (hier 5) an eine Funktion

void AddiereArrays(int ia[], int ib[], ic[], int Arraylaenge)

Die Funktion soll die Werte aus den Arrays ia[i] und ib[i] adieren und in ic[i], speichern.

2. Schreiben Sie eine Funktion

PrintArray(int ia[], int Arraylaenge]

die die Werte des Arrays auf dem Bildschirm ausgibt.

Strings

Die ersten Aufgaben können Sie mit Hilfe von statischen Arrays durchführen. Sobald sie jedoch Pointer in der Vorlesung hatten, ist es meist sinnvoller oder sogar notwendig, die Funktionen mit Pointern zu programmieren.

Aus Übungszwecken sollten Sie versuchen alle Aufgaben, die Sie mit Arrays programmiert haben, auch mit Hilfe von Pointern programmieren.

1.) Schreiben Sie eine Funktion, die die Länge eines Strings zurückgibt
2.) Schreiben Sie eine Funktion, die zählt, wie oft ein Buchstabe in einem String vorhanden ist
3.) Schreiben Sie eine Funktion, die alle Vokale (a,e,i,o,u) zählt
4.) Schreiben Sie eine Funktion, die einen bestimmten Buchstaben sucht und durch einen anderen ersetzt
5.) Schreiben Sie eine Funktion, die alle Großbuchstaben in Kleinbuchstaben und alle Kleinbuchstaben in Großbuchstaben umwandelt (schwer !)
6.) Schreiben Sie eine Funktion, die zwei übergebene Strings hintereinander in ein neu mit malloc erzeugten String kopiert und den Pointer zurückgibt
7.) Schreiben sie eine Funktion, die prüft, ob ein String in einem anderen String enthalten ist.
8.) Schreibe eine Funktion, die zwei Strings miteinander vergleicht. Unterscheiden sich im zweiten String Zeichen an der gleichen Position im ersten String, werden diese in einem Ergebnis-String gespeichert. Das Problem ist mit einer Schleife und einer **if** Anweisung lösbar. Rückgabewert ist der Zeiger auf den Ergebnis String, also Parameter **result**. Sind die Strings identisch, wird ein Leer-String zurückgegeben. Der Funktions-Prototyp und Beispiel-Ausgabe:

```
char * stringCompareDif(char *string1, char *string2, char *result);

String 1:     Die schoenen Dinge im Leben sind umsonst.
String 2:     die Schoenen dinge Im leben Sind Umsonst.
Unterschiede:  dSdIlSU
```

9.) Schreibe eine Funktion, die in einem String bestimmte Zeichen entfernt (durch Leerzeichen ersetzt). Der erste Parameter der Funktion ist der zu bearbeitende String. Der zweite Parameter enthält die zu entfernenden Zeichen.

```
char * stringRemoveChars(char *string, char *spanset);

vorher:      8Die1schoenen4Dinge6im3Leben9sind0umsonst.5
nachher:       Die schoenen Dinge im Leben sind umsonst.
```

10.) Es soll eine Funktion geschrieben werden, welche in einem String einen Teilstring ersetzt. Der gesuchte String und der Ersetzungsstring können hierbei eine unterschiedliche Länge haben. Dafür wäre die Verwendung der dynamischen Speicherverwaltung hilfreich.

char * stringReplace(char *search, char *replace, char *string);

vorher: Die schoenen Sachen im Leben sind umsonst.
nachher: Die schoenen Dinge im Leben sind umsonst.

11.) String Rotation: schreiben Sie eine Funktion, die den ersten Buchstaben eines Strings an die letzte Stelle schreibt und alle anderen eine Stelle nach vorne holt.

Hallo
alloH

12.) modifizieren Sie die Funktion, so dass sie den String vorwärts und rückwärts rotieren können

vorwärts: Hallo -> alloH
rückwärts. alloH ->Hallo

13.)Anschließend implementieren Sie eine Schleife im Hauptprogramm, das die Funktion 20x mit zwei Strings aufruft, je einmal vorwärts und einmal rückwärts. Nach jedem Aufruf (der Funktion mit beiden Strings) geben Sie die beiden Strings ohne Leerzeichen hintereinander auf dem Bildschirm aus:

st1: Hallo
st2: ollaH

HalloollaH
alloHHolla
lloHaaHoll
..

14.) Schreiben sie eine Funktion, die die Anzahl der Vokale zählt. Dabei soll sie sich über alle Aufrufe hinweg merken, wie oft jeder Vokal vorgekommen ist. Immer wenn ein Vokal 10mal vorgekommen ist, soll sie HURRA und den entsprechenden Vokal auf dem Bildschirm schreiben. Rufen sie die Funktion mehrfach im Hauptprogramm mit passenden Strings auf.

15.) Modifizieren Sie die Funktion so, dass sie

a.) die Anzahl der Vokale zählt
b.) die Anzahl eines bestimmten Vokals auf den Bildschirm ausgibt. Der Vokal soll durch einen Parameter bestimmbar sein.

Structs

1. deklarieren sie einen struct für eine Adresse, vermerken Sie in dem struct, ob die Adresse benutzt wird
2. definieren Sie ein Array mit 100 Adressen in der main funktion
3. schreiben Sie eine Funktion zum setzen der Werte des i'ten Eintrags in dem Array

Achtung: Strings können nicht zugewiesen werden !! sie müssen char by char in ein array kopiert werden

4. setzen Sie sinnvolle Werte für verschieden Adressen ein
5. schreiben Sie eine Funktion zur Ausgabe des i'ten Eintrags des Arrays
6. schreiben Sie eine Funktion zur Ausgabe aller Einträge des Arrays
7. schreiben Sie eine Funktion die alle Adressen zu einer PLZ ausgibt
8. schreiben Sie eine Funktion, die alle Adressen mit einer bestimmten PLZ löscht
9. Schreiben Sie eine Funktion, die die i'te Adresse der Liste zurück gibt
10. Schreiben Sie eine Funktion, die die Anzahl der Adressen zählt
11. Schreiben Sie eine Funktion, die prüft, ob die i'te Adesse benutzt wird
12. Schreiben Sie eine Funktion, die zwei existierende Adressen vertauscht
13. Schreiben Sie eine Funktion, die alle Adressen löscht

Listen

1. Definieren Sie sich ein passendes struct für eine lineare Liste.
2. Schreiben Sie eine Funktion, die eine leere Liste erzeugt (Kopfelement)
3. Schreiben Sie eine Funktion, die ein neues Listen-Element erzeugt und AM ANFANG der Liste einfügt
4. Schreiben Sie eine Funktion, die ein neues Listen-Element erzeugt und AM ENDE der Liste einfügt
5. Schreiben Sie eine Funktion, die die Werte eines Elements auf dem Bildschirm ausgibt
6. Schreiben Sie eine Funktion, die alle Elemente auf dem Bildschirm ausgibt
7. Schreiben Sie eine Funktion, die jeweils zwei Elemente der Liste miteinander vergleicht und in eine Reihenfolge bringt.
8. Schreiben Sie eine Funktion, die das kleinste Element in der Liste findet und geben sie den Wert zurück
9. Schreiben Sie eine Funktion, die das erste Element in der Liste mit einem bestimmten Wert findet und dies aus der Liste löscht
10. Schreiben Sie eine Funktion, die prüft, ob ihre Liste leer ist
11. Nutzen Sie die oben beschriebenen Funktionen, um aus einer existierenden Liste eine neue - jetzt aber aufsteigend sortierte Liste erstellt. Die alte Liste ist danach leer.
12. wie zuvor, nur absteigend sortiert
13. Modifizieren Sie die beiden Funktionen zum Hinzufügen von Elementen in die Liste so, dass keine gleichen Werte in der Liste enthalten sein können (bei doppelten Werten kein neues Element hinzufügen)
14. Schreiben Sie eine Funktion, die den Element Pointer auf das Vorgänger Element in der Liste zu einem vorgegebenen Wert zurückgibt
15. Schreiben Sie eine Funktion, die zu zwei vorgegebenen Werten die Elemente findet und diese in der Liste vertauscht (nicht die Werte vertauschen!)
16. Erweitern Sie die Struktur, so dass sie einen Vornamen und einen Nachnamen enthält
17. Modifizieren Sie alle Funktionen derart, dass passende Werte für Vorname und Name angegeben und verwendet werden können (Achtung: die Strings müssen dynamisch mit malloc und free verwaltet werden)
18. Nutzen Sie die Funktionen, um den Bubblesort Algorithmus zu implementieren
19. Schreiben Sie eine Funktion, die alle Listenelemente mit einem vorgegebenen Namen findet und diese auf dem Bildschirm ausgibt

20. Programmieren Sie mit Hilfe der Listenfunktionen einen Stack (Stapel) mit den Funktionen pushStack (Element auf den stapel legen), popStack (Element vom Stapel nehmen, den Inhalt mit printf auf dem Bildschirm ausgeben), isStackempty (testet ob der Stapel leer ist) und printStack (gibt die Elemente des Stapels auf dem Bildschirm aus)
21. Sortieren Sie die Liste nach dem Namen (Umsortieren der Elemente)