1 Testataufgaben

1.1 Projekt (Woche 1) **(3 Punkte)**

In den letzten acht Wochen haben Sie die Grundlagen der prozeduralen Programmierung in der Programmiersprache `C` gelernt und schon viele kleinere Programme entwickelt. Nun ist es an der Zeit, sich an einem eigenen größeren Programm zu versuchen. Ihre Aufgabe ist es, in den Gruppen innerhalb von drei Wochen ein Projekt zu planen, umzusetzen und zu testen. Was Ihr Programm tun soll, ist Ihnen überlassen, egal ob Mediathek-Verwaltung oder Spiel oder etwas vollkommen anderes. Die Aufgabenverteilung für die drei Wochen sieht wie folgt aus:

Woche 1	Planung des Projekts	(3 Punkte)
Woche 2	Implementierung	(7 Punkte)
Woche 3	Implementierung, Test, Präsentation	(7 Punkte)

Begleitet wird das Ganze von kleineren Übungsaufgaben zur Vorlesung. Innerhalb der nächsten Woche soll die schriftliche Planung des Projekts erfolgen, **die Sie Ihrem Tutor im Testat spätestens am 14.01.2021 abgeben**. In der Planung sollen enthalten sein:

(A) **Spezifikation**

Eine ausführliche Beschreibung der Funktionalität. Nach der Fertigstellung werden Sie danach bewertet, inwieweit die Implementierung mit dieser Beschreibung übereinstimmt. Stellen Sie dafür mehrere (ca. 8) wesentliche Leistungsmerkmale auf, nach denen Ihr Programm bewertet werden soll.

(B) **Design**

Eine detaillierte Beschreibung der von Ihnen angestrebten technischen Umsetzung der in (A) spezifizierten Funktionalitäten:

(a) Programmstruktur/Aufbau des Programms

(b) wichtigste Funktionen und ihre Signatur (Parameter und Rückgabewert)

(c) zentrale (strukturierte) Datentypen

1.2 Sortieralgorithmen **(7 Punkte)**

Es ist ein Rahmenprogramm vorgegeben, das dem Benutzer folgendes Auswahlmenü in der Konsole anzeigt:

```
1) Array-Groesse festlegen        4) Bubblesort (downwards)
2) Zahlen von Tastatur einlesen   5) Mergesort
3) Zufallszahlen erzeugen         6) Programm beenden
```

Mit den implementierten Funktionen lassen sich zur Programmlaufzeit die Länge N der zu sortierenden Elemente festlegen, Zahlen (1-1000) einlesen oder zufällig erzeugen und anschließend sortieren.

Schreiben Sie zwei Funktionen für Sortieralgorithmen[1] in einer separaten Quelldatei (z.B. `sort.c`), inkludieren Sie im Rahmenprogramm einen zugehörigen Header (z.B. `sort.h`) mit den Funktionssignaturen aus der separaten Quelldatei und fügen Sie die Funktionsaufrufe an entsprechender Stelle ein. Beide Funktionen sortieren jeweils ein Array der Länge N von ganzen Zahlen in aufsteigender Reihenfolge und geben es anschließend in der Konsole aus:

1. Bubblesort (downwards): Mit zwei ineinander geschachtelten Schleifen soll durch bedingtes Vertauschen benachbarter Werte in der inneren Schleife erreicht werden, dass nach dem i-ten äußeren Schleifendurchlauf die i kleinsten Elemente in aufsteigender Reihenfolge an den ersten i Stellen des Arrays stehen.

2. Mergesort: Ein Array wird, solange $N > 1$ gilt, in ein linkes und rechtes Teilarray aufgeteilt. In jeweils einem rekursiven Funktionsaufruf werden beide Teilarrays sortiert. Anschließend werden beide sortierten Teilarrays zu einem sortierten Array wieder zusammengefügt.

[1] `http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/022_c_algorithmen_003.htm`

2 Präsenzaufgaben

1. Schreiben Sie eine rekursive Funktion `instring`, die zwei Strings `str1` und `str2` als einzige Eingabeparameter besitzt und testet, ob `str1` ein Teilstring von `str2` ist. In diesem Fall soll die Anfangsposition von `str1` in `str2` zurückgegeben werden. Wenn `str1` kein Teilstring von `str2` ist, soll -1 zurückgegeben werden.

   Eine kurze Implementation ist durch folgende **rekursive** Lösung möglich:

   (a) Ein leerer String `""` ist Anfangsstring eines jeden anderen Strings.

   (b) Ein nichtleerer String ist kein Teilstring eines leeren Strings.

   (c) Für einen nichtleeren String `str` bezeichne `str_+` den Reststring von `str` nach dem ersten Buchstaben. Wenn nun die Anfangsbuchstaben von `str1` und `str2` übereinstimmen, so ist `str1` genau dann ein Anfangsstring von `str2`, wenn `str1_+` ein Anfangsstring von `str2_+` ist.

   (d) Wenn andererseits die Anfangsbuchstaben von `str1` und `str2` nicht übereinstimmen und `str1` ein Teilstring von `str2_+` ist, so lässt sich die Startposition von `str1` in `str2` leicht aus der Startposition von `str1` in `str2_+` ermitteln. Falls `str1` kein Teilstring von `str2_+` ist, so ist das Ergebnis ebenfalls klar.

   Beispiel: Der Aufruf `instring("bahn","Eisenbahn")` soll das Ergebnis "5" liefern und der Aufruf `instring("Bahn","Eisenbahn")` das Ergebnis "-1".

2. Ein einfaches Programm, soll das „Paradoxon des Wettrennens zwischen Achilles und der Schildkröte"[2] widerlegen. Es gelten folgende Annahmen: Die ersten 500 Meter läuft Achilles mit 20 $\frac{m}{s}$, dann schafft er nur noch 5 $\frac{m}{s}$. Die Schildkröte kriecht dagegen konstant mit 5 $\frac{cm}{s}$. Da die Schildkröte viel langsamer ist, bekommt sie einen Vorsprung von 1000 Metern. Schreiben Sie im ersten Schritt eine Funktion mit der Signatur

   ```
   int zeit(float xA, float xS, int t)
   ```

   die den Überholzeitpunkt von Achilles **rekursiv** berechnet und zurück gibt. Zur Lösung der Aufgabe, rufen Sie die Funktion so lange rekursiv auf, wie die Position des Achilles `xA` kleiner als die der Schildkröte `xS` ist. Aktualisieren Sie dabei vor jedem rekursiven Aufruf die Position des Achilles, die der Schildkröte und den Sekundenzähler `t` entsprechend den Angaben von oben.

   Schreiben Sie zusätzlich ein Hauptprogramm, in dem Sie die Funktion `zeit` mit korrekten Startwerten aufrufen und die berechnete Überholzeit am Bildschirm ausgeben.

[2] Ein antiker griechischer Philosoph, Zenon von Elea, behauptete, dass auch ein so schneller Läufer wie Achilles eine Schildkröte nicht einholen kann, wenn man der Schildkröte einen Vorsprung lässt.

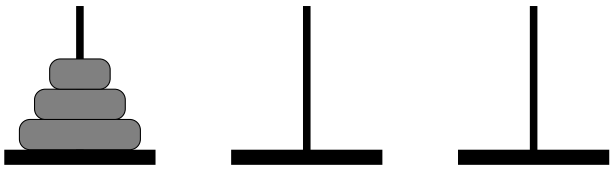
3. Schreiben Sie ein vollständiges Programm, das zunächst eine positive ganze Zahl $n > 0$ von der Tastatur einliest. Dann alle Wörter der Länge n, die nur aus den beiden Buchstaben `'a'` und `'b'` bestehen, übersichtlich ausgibt.

   Beispiel für $n = 3$: `aaa,baa,aba,bba,aab,bab,abb,bbb`.

   Hinweis: Bei einer Lösung mittels einer rekursiven Funktion kann es hilfreich sein, einen String-Parameter zum Zwischenspeichern von Teilwörtern zu verwenden.

4. Ein alter Mönch bekam die Aufgabe, einen Scheibenturm, bestehend aus mehreren Scheiben, von einer Startposition an eine Zielposition unter folgenden Bedingungen zu transportieren:

   - Es darf nur die oberste Scheibe eines Turms genommen werden.
   - Es darf nie eine größere Scheibe auf einer kleineren liegen.
   - Es darf eine Hilfsposition zum Zwischenlagern verwendet werden.

   Der Mönch schrieb folgende **rekursive** Lösung an seine Tür:

   (a) Besteht der Turm aus $n > 1$ Scheiben, dann transportiere den Teilturm der Höhe $n-1$ von der Startposition zur Hilfsposition unter Zuhilfenahme der Zielposition als temporärer Hilfsposition.

   (b) Verschiebe nun die verbliebene (unterste) Scheibe von der Start- zur Zielposition.

   (c) Anschließend transportiere den Teilturm der Höhe $n-1$ von der Hilfsposition zur Zielposition unter Zuhilfenahme der Startposition als temporärer Hilfsposition.

   Implementieren Sie den Algorithmus in einer **rekursiven** Funktion

   ```
   void verschiebe(int start, int ziel, int hilf, int n)
   ```

   Die Funktion soll zu vorgegebener Start- und Endposition und vorgegebener Anzahl der Scheiben die Transportreihenfolge der Scheiben am Bildschirm ausgeben.