

Allgemeine Hinweise:

Ab diesem Übungsblatt dürfen Sie nur noch die Techniken der prozeduralen Programmierung verwenden. Der Header `fcpp.hh` darf nur noch für die Ein- und Ausgabe von Zahlen verwendet werden.

Diese Aufgaben beschäftigen sich in erster Linie mit Feldern (Arrays). Bitte beachten Sie das Folgende:

- Schreiben Sie keine Funktionen, welche ein Array als Argument oder als Rückgabotyp haben.
- Kopieren Sie Arrays elementweise. Statt `int a[n] = b;` schreiben Sie

```
int a[n];
for(int i=0; i<n; i++)
    a[i] = b[i];
```

Sie werden in Kürze lernen, warum wir Ihnen diese Einschränkungen empfehlen!

Bitte achten Sie von diesem Aufgaben an darauf, dass Ihre Programme übersichtlich formatiert sind. Dies bedeutet unter anderem:

- Verwenden Sie sprechende Variablenamen! Variablenamen aus nur einem Buchstaben sind nur für Schleifenzähler akzeptabel.
- Rücken Sie Blöcke (im Sinne von C++) einheitlich ein.
- Definieren Sie Variablen erst in der Umgebung, in der sie gebraucht werden, um die Namensräume möglichst klein zu halten.
- Kommentieren Sie Ihr Programm!

Übung 1 *Schleifeninvariante Fibonacci*

In Abschnitt 3.8 des Vorlesungsskripts haben Sie das Aufstellen und Nachweisen der Schleifeninvariante für die prozedurale Berechnung der Fakultätsfunktion mittels einer `while`-Schleife kennengelernt. Sie sollen nun analog für die Berechnung der Fibonacci-Zahlen mittels einer `for`-Schleife (Abschnitt 3.6) vorgehen.

Damit die Variable `i` auch schon vor dem ersten Schleifendurchlauf definiert ist, verwenden Sie diese leicht abgewandelte Version des Algorithmus:

```
int fib ( int n)
{
    int a=0;
    int b=1;
    int i=0;
    for (i=0; i<n ; i=i+1)
    {
        int t=a+b ; a = b ; b = t ;
    }
    return a ;
}
```

1. Sie haben gelernt, dass `for`-Schleifen äquivalente Konstrukte zu `while`-Schleifen sind. Wie lauten die Variablenbelegungen v , die Schleifenbedingung $B(v)$ sowie der Schleifentransformator $H(v)$, wenn Sie die gegebene `for`-Schleife in die Form der kanonischen `while`-Schleife

```
while ( B(v) ) { v=H(v); }
```

transformieren?

2. Stellen Sie nun die Schleifeninvariante $INV(v)$ auf und weisen Sie deren partielle Korrektheit nach. Geben Sie dazu auch Vor- und Nachbedingungen $P(n)$ und $Q(n)$ des Algorithmus an.

(5 Punkte)

Übung 2 Zahlen in Array einsortieren

Schreiben Sie ein Programm, das natürliche Zahlen von der Standardeingabe liest. Sie können dazu die Funktion `enter_int` aus dem Header `fcpp.hh` verwenden. Die eingelesenen Zahlen sollen in ein Array der Größe 10 geschrieben werden, und zwar derart, dass dieses Array stets aufsteigend sortiert bleibt.

Bei Eingabe von -1 soll das Programm terminieren, ansonsten werden so lange weitere Zahlen eingelesen, bis das Array voll ist. Der Versuch, in ein volles Array zu schreiben, soll das Programm mit einer Fehlermeldung beenden. Bei Eingabe einer 0 soll das gesamte Array ausgegeben werden. Die Ausgabe des Arrays könnte dabei wie folgt aussehen:

```
Array = [6 7 8 9 10 0 0 0 0 0]
```

Fehleingaben (negative Zahlen außer der -1) sollen erkannt und das Programm mit einer Fehlermeldung beendet werden.

(5 Punkte)

Übung 3 Perfect Shuffle

Beim Mischen eines Kartenblatts gibt es unter versierten Pokerspielern und Zauberkünstlern die Technik des sogenannten 'Perfect Shuffle'. Dabei wird das Kartenblatt in zwei exakt gleich große Teile geteilt und so gemischt, dass immer abwechselnd eine Karte von jedem dieser Stapel genommen wird. Dabei kann man zwei Vorgehensweisen unterscheiden:

- Perfect-Out-Shuffle: $ABCDEFGH \Rightarrow AEBFCGDH$ (oberste Karte bleibt oben)
- Perfect-In-Shuffle: $ABCDEFGH \Rightarrow EAFBGCHD$ (oberste Karte wandert)

Ihre Aufgabe ist es, ein Programm zu schreiben, das diese beiden Mischmethoden simuliert um die folgenden Frage zu beantworten: Wie oft muss man das Kartenblatt bei ausschließlicher Verwendung von Perfect-In bzw. Perfect-Out mischen um wieder in den Ausgangszustand zu gelangen?

Halten Sie sich dafür an die folgenden Hinweise:

- Verwenden Sie als Datenstruktur für das Deck ein Array von Integerwerten. Die Einträge des Arrays kodieren dabei die Kartenwerte. Das Deck wird mit den Werten $0, \dots, n-1$ initialisiert.
- Schreiben Sie eine Funktion, die überprüft ob sich das Deck im Ausgangszustand befindet. Ausnahmsweise gilt hier der Hinweis zu Beginn, Arrays nicht als Funktionsargumente zu verwenden, nicht. Verwenden Sie die folgende Signatur für diese Funktion:

```
bool deck_check(int deck[], int n)
```

Dabei ist `deck` die Variable, die das Kartenblatt beschreibt und `n` dessen Größe.

- Ermitteln Sie die Anzahl der Mischvorgänge, die benötigt wird, bis das Deck wieder ungemischt (also im Ausgangszustand) ist, durch Verwendung einer Schleife in der `main`-Funktion.

(10 Punkte)