UNIVERSITÄT HEIDELBERG
Zukunft. Seit 1386.

IWR
Interdisciplinary Center for Scientific Computing

IPI: Einführung in die Praktische Informatik
Wintersemester 2018
Prof. Björn Ommer
ommer@uni-heidelberg.de

Übungsblatt 6: Prozedurale Programmierung und Felder

Abgabe am 05.12.2018, 13:00.

Hinweis 1: Beginnend mit diesem Blatt dürfen Sie die Techniken der *prozeduralen Programmierung* verwenden, sofern diese in der Vorlesung behandelt wurden. Verzichten Sie daher vor allem auf Rekursion und die `cond`-Funktion und lösen Sie die gestellten Aufgaben durch die Verwendung von Schleifen und bedingten Anweisungen.

Hinweis 2: Bitte achten Sie von diesem Zettel an darauf, dass Ihre Programme übersichtlich formatiert sind. Dies bedeutet unter anderem:

- Verwenden Sie selbsterklärende Variablennamen! Variablennamen 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 auch gebraucht werden, um die Namensräume möglichst klein zu halten.
- Kommentieren Sie Ihr Programm!

Beachten Sie, dass wie in der Vorlesung angekündigt, grobe Verstöße gegen diese Regeln ab dieser Woche zu Punktabzug führen kann!

Aufgabe 1: Methode der Intervallhalbierung zur Nullstellensuche (5P)
Sie haben das Newtonverfahren zur Bestimmung von Quadratwurzeln in der Vorlesung kennengelernt. Eine andere Methode ergibt sich aus der Methode der Intervallhalbierung. Diese Methode wird eigentlich zur Bestimmung einer Nullstelle einer stetigen Funktion $f(x)$ verwendet. Wenn man aber $f(x) = x^2 - w$ ansetzt, erhält man dadurch $\sqrt{w}$.
Die grundlegende Idee der Intervallhalbierung ist: Wenn a und b gegeben sind und $f(a) < 0 < f(b)$ gilt, dann muss mindestens eine Nullstelle zwischen a und b liegen. Um die Nullstelle zu finden, berechnet man den Mittelwert m von a und b. Wenn $f(m) > 0$ gilt, so muss f eine Nullstelle zwischen a und m haben. Wenn $f(m) < 0$ gilt, so muss f eine Nullstelle zwischen m und b haben. Durch wiederholtes Anwenden wird das Intervall, in dem der gesuchte Wert liegt, immer kleiner.

a) Implementieren Sie dieses Verfahren analog zu dem Programm `newton.cc` aus der Vorlesung, d.h. schreiben Sie Funktionen analog zu `double wurzelIter(double xn, double a)` und `double wurzel(double a)`. [3 Punkte]

b) Bestimmen Sie experimentell, wie viele Iterationen die Intervallhalbierung braucht, um die Werte für $\sqrt{2}$, $\sqrt{3}$ und $\sqrt{4}$ bis auf eine Genauigkeit von 10^{-12} zu berechnen. Beginnen Sie mit den Startwerten 0 und 4. Vergleichen Sie das mit dem Newtonverfahren. [2 Punkte]

Aufgabe 2: Schleifeninvariante Fibonacci (5P)

In der Vorlesung 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 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;
}
```

a) 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? [1 Punkt]

b) 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. [4 Punkte]

Aufgabe 3: Zahlen in Array einsortieren (5P)

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

Aufgabe 4: Perfect Shuffle (5P)

Beim Mischen eines Kartenblatts gibt es unter versierten Pokerspielern und Zauberkünstlern die Technik des sogenannten 'Perfect Shuffle'. Dabei wird das Kartenblatt (bestehend aus 52 Karten) 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 $0, \ldots, n-1$ initialisiert.
- Schreiben Sie eine Funktion, die überprüft ob sich das Deck im Ausgangszustand befindet. Hier gilt die obige Bemerkung, Arrays nicht als Funktionsargumente zu verwenden, ausnahmsweise 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.