Prof. Dr. A.K. Hartmann
WiSe 2023/24, 5.04.4521
Blatt 3, 19.02.2024

Übungen zur Computerorientierten Physik

3 Array Permutationen 2

In dieser Übung sollen Sie rekursive Funktionen üben und Eigenschaften vor Permutationen bestimmen.

- Laden Sie das Programm `permutation_fragment.c` von StudIP. Schauen Sie sich das vorhandene Material an.
- Entwerfen, implementieren und testen Sie eine *sequenzielle* Funktion, die für ein Array `a[]` ermittelt, für wie viele der benachbarten Paare `a[t],a[t+1]` das erste Element kleiner ist, das Array dort also "up" läuft.

  Der Funktionsprototyp sieht wie folgt aus:

```
/********************** count_up() ************************/
/** Counts how often an array element is followed by a     **/
/** larger element.                                        **/
/**                                                        **/
/** Parameters: (*) = return parameter                     **/
/**        n_max: size of array                            **/
/**            a: array                                    **/
/** Returns:                                               **/
/**        number of 'up' pairs                            **/
/************************************************************/
int count_up(int n_max, int *a)
```

(2 P)

- Entwerfen, implementieren und testen Sie eine *rekursive* Funktion `permutation()`, die für ein Array `a[]` von Integer-Zahlen alle Permutationen erzeugt und ausgibt. Hinweis: man kann die Funktion so schreiben, so dass die Permutationen am Platz generiert werden, man braucht also kein zusätzliches Array.

  Der Funktionsprototyp sieht wie folgt aus:

```
/****************** permutation() ************************/
/** Obtains all permutations of positions 0..n-1 of a     **/
/** given array 'a' of numbers and prints them if n==1,   **/
```

```
/** including the higher index entries (from 0..n_max-1). **/
/** Also a statistics on the permutations regarding       **/
/** 'up_count()' is performed                             **/
/**                                                       **/
/** Parameters: (*) = return parameter                    **/
/**            n: current range                           **/
/**        n_max: size of array                           **/
/**            a: array                                   **/
/**      (*)  up: pointer to total number of 'up' pairs   **/
/**      (*) num: pointer to number of permutations       **/
/** Returns:                                              **/
/**        (nothing)                                      **/
/*******************************************************/
void permutation(int n, int n_max, int *a, double *up, double *num)
```

Grundidee: Um das Problem für die Elemente `0..(n-1)` zu lösen (anfangs `n`=`n_max`), setzt man in das letzte Element `a[n-1]` durch Tauschen alle Werte aus den Elementen `0...n-1` und ruft jedes mal die Funktion für die Elemente `0..(n-2)` auf.

Sie können das vorgegebene Hauptprogramm verwenden, das ein Array mit den Zahlen 0 bis `n-1` füllt und `permutation()` aufruft.

Die Funktion und das Hauptprogramm sind in `permutation_fragment.c` schon darauf vorbereitet die kleine "up" Statistik zu errstellen.

Tipps:

- Vergessen Sie nicht die Nummern zurück an ihre richtigen Stellen zurück zu tauschen (am besten nach jedem Aufruf).
- Bei der Ausgabe geben Sie jeweils alle `n_max` Element aus.

(6 P)

- Messung:

  Lassen Sie das Programm für Zahlen $n = 2, 3, \ldots, 10$ laufen. Wie entwickelt sich die Zahl der Permutationen? Wie entwickelt sich die mittlere Zahl der "up" Paare und warum muss das auch so sein? (2 P)