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

Exercises Computational Physics

3 Array Permutations 2

In this exercise you shall implement a recursive function and learn about the properties of permutations.

- Download the program `permutation_fragment.c` from StudIP. Investigate the available code.

- Design, implement and test a *sequential* function, which counts for an array `a[]` the number of neighboring pairs `a[t],a[t+1]` for which the first element is smaller, i.e., the array runs locally "up".

  The function prototype looks as follows:

```
/********************** 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)

- Design, implement and test a *recursive* function `permutation()`, which creates for an array `a[]` of integer numbers all permutationen, and prints them. Hint: you can write the function such that the permutations are generated in place, i.e., you do not need a second array (but you are allowed to use one).

  The function prototype looks as follows:

```
/****************** 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)
```

Basic idea: To solve the problem for elements `0..(n-1)` (initially `n`=`n_max`), one iteratively assigns the last element `a[n-1]` (by exchanging) to all possible array elements `0...n-1` and the calls the function for the remaining elements `0..(n-2)`.

You can use the given main function which creates an array with numbers 0 to `n-1` and calls `permutation()`.

The function and the main function in `permutation_fragment.c` are prepared to do the “up” statistics.

Hints:

- Do not forget to put the exchanged numbers back to the original places (ideally within each iteration).
- When printing the permutations, show all `n_max` elements.

(6 P)

- Measurement:

  Execute the program for $n = 2, 3, \ldots, 10$. How does the number of permutationen behave? What do you get for the average number of “up” pairs. Can you explain it? (2 P)