High Performance Computing
Exercise Sheet 7

HS 25
Dr. Douglas Potter

http://www.astro.uzh.ch/
Teaching Assistants:
Cesare Cozza, cesare.cozza@uzh.ch
Mark Eberlein, mark.eberlein@uzh.ch

Issued: 31.10.2025
Due: 07.11.2025

In this exercise sheet, we will practice some pointer basics from C/C++ programming. This is essential to understand when programming in MPI.
Remeber to submit your solution to Exercise 3 of Sheet 6.

Exercise 1 [Pointers]

Pointer are used extensively both in C and C++. Pointers are variables storing an address of some object (or other variable). Let us define a pointer `p` to be an integer array with length of 10 with the following values.

p	10	20	30	40	50	60	70	80	90	100

Answer the following questions:

a. What C/C++ **type** is `p`? → Array of Integers with length 10 int[10]
b. What is the **type** of `*p`? → int
c. What is the **type** of `&p`? → pointer to array
d. What is the **value** of `*p`? 10
e. What is the **value** of `p[0]`? 10
f. What is the **value** of `p[3]`? 40
g. What is the **value** of `*(p+5)`? 60
h. What is the **value** of `*p+5`? 15
i. In some languages (e.g. Python), `p[-1] = 42` sets the last (10th) element to 42. What does `p[-1] = 42` mean in C/C++?
→ not possible → tries to access memory before array

j. Let: `int *q = p-1;` → invalid memory access
What is the **value** of `q[0]`. Careful!

k. What is the **value** of `*(q+10)`? → *(p+9) = 100

l. What is the **value** of `*(q+11)`? → *(p+10) → invalid memory access

Exercise 2 [More Pointers]

Consider the following C++ code.

```
int ** M = new int *[2];
M[0] = new int[5]{0,1,2,3,4};
M[1] = new int[5]{5,6,7,8,9};
```

if the above does not work with your compiler, you can simply use:

```
int** M = new int *[2];
M[0] = new int[5];
M[1] = new int[5];

for (int i=0;i<5;i++){
  M[0][i] = i;
  M[1][i] = 5+i;
}
```

Answer the following questions:

a. What **data type** is `M` (integer, pointer to integer, ...)? pointer → pointer → int

b. What are `M`, `*M`, `**M`? → P to P, P to Int, int

c. What is `M[1][3]`, `*(M[0]+1)`, `*(*(M+1)+3)`? M[1][3] = 8, *(M[0]+1) = 1, *(*(M+1)) = 8

d. Are the `M[0]` and `M[1]` arrays stored next to each other in the memory? In other words, is it true that `*(M[0]+5)==M[1][0]`? → failed
→ We can not easily controll the "actual" location in memory

Write a short code that would initialize the array `M` as above and code a function that will swap the array along axis 0, thus the result should be `M2`, where `M2[0]` contains vector `[9,8,7,6,5]` and `M2[1]` vector `[4,3,2,1,0]`. In your code, you are NOT ALLOWED to access the values using bracket notation (e.g. M[1][2]) but should instead use * to get the needed values.

Commit: Push your code to the repository.

Exercise 3 [Array strides]

In class we learned that 2D arrays (or more generally n-dimensional arrays) are stored contiguously in memory in either row-major or column-major order. Consider a 2D array with x and y coordinates with 14 elements along x and 6 elements along y. We use the notation (row index, column index).

y														
5	5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9	5,10	5,11	5,12	5,13
4	4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9	4,10	4,11	4,12	4,13
3	3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9	3,10	3,11	3,12	3,13
2	2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,10	2,11	2,12	2,13
1	1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,10	1,11	1,12	1,13
0	0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,10	0,11	0,12	0,13
	0	1	2	3	4	5	6	7	8	9	10	11	12	13

x

Answer the following questions.

a. Which variable (x or y) tells you which row it is, and which tells you the column? Row = y Col = x

b. How many total elements are in the table? 6 × 14 = 84

c. Are the elements $(1,3)$ and $(1,4)$ adjacent in memory, or is it rather $(1,3)$ and $(2,3)$? → rows major

d. Is one of these choices "better"? Why or why not? → if i continuesly acces adjecent row elmts → row major is faster & more cache-friendly

e. From now on assume "row-major" order, meaning that elements in each row are adjacent to each other. The "stride" tells you the distance between elements in the given direction. What is the stride for the x dimension? 1

f. What is the stride of the y dimension? 14

g. Remember again that elements are stored adjacent in memory. Assume that rows are stored adjacent to each other. This means that if you have a table of integers (`int *A`) then the element $(0,0)$ is at $A[0]$, element $(0,1)$ is at $A[1]$ and so on. Where is element $(0,10)$? A[10]

h. Where is element $(3,0)$?

A[42]

i. Where is element $(7, 4)$? -> no col 7? -> out of bound error

Write a function `ii(x,y)` that takes x and y and returns the location of the element in `A`. You function will look like this:

```
// Given an x and y coordinate, return the
// index into an array of 14 columns by 6 rows
int ii(int x,int y) {
    return ...
}
```

You will need to use both index variables and their strides. Write a test program similar to the following to verify your results.

```
int main() {
    int x,y;
    for(x=0; x<14; ++x) {
      for(y=0; y<6; ++y) {
        printf("(%d,%d) %d\n",x,y,ii(x,y));
      }
    }
```

Commit: Push your code to the repository.

Reminder

Friendly reminder to upload your solution for the MPI Poisson solver (Exercise 3 on Sheet 6).