Distance and Neighbouring Points:

Intro : Modern vehicles (cars, industrial trucks, planes, robots, ...) have camera, infrared, laser, radar, ... sensors to continuously scan obstacles in the direction of movement. Their data are needed for distance, collision avoidance, breaking assistance, ... systems. In this task a simplified version of obstacles shall be modeled by points detected by an imaginary sensor system placed in the front and middle of the vehicle as origin of a relative local two-dimensional cartesian coordinate system. Arbitrary many of such obstacles/points in front of a vehicle shall be stored in a list. This list shall be sorted by the Euclidian distance from the origin (see picure above and examples below).

Task 1 : To simply identify them each obstacle/point shall store a string ("A", "B", ... in example above), its coordinate (x, y) in the local coordinate system and its distance to the origin. Write a type definition for a structure with these four data as well as a pointer to the next list element as components/variables of the structure.

Task 2 : Write a function to calculate the Euclidian distance dd of two obstacles/points P1 and P2 with coordinates (x1,y1)(x1,y1) and (x2,y2)(x2,y2) by the formula: $d=\sqrt{(x1-x2)2+(y1-y2)2}$ and return dd as function value.

Task 3: Write a function with a list of obstacles/points as first and one obstacle/point as second parameter calculating the nearest other obstacle/point to it and returning it as pointer. Take care that an obstacle/point does not return itself (distance 0) as well as that at least two obstacles/points need to exist, otherwise a null pointer nullptr shall be returned.

Task 4: Write a function with a list of obstacles/points sorted by distance from the origin of the coordinate system as first parameter and a pointer to a new obstacle/point as second parameter. In its body insert sorted this new obstacle/point into the list and return a pointer to the (new) head of the list of obstacles/points.

Task 5: Write a function to output formatted the list of obstacles/points (all numbers with six characters and two digits after decimal point). Beside the identifying string of the obstacle/point, its coordinates and its distance additionally the nearest neighbor obstacle/point to it shall be found and outputted (for each call the function from subtask 3; see example below).

Task 6 : Write a function to delete the whole list of all obstacles/points completely on the heap.

Task 7 : In a loop in function main arbitrary many of such obstacles/points shall be inputted and stored in a list sorted by distance in ascending order - call for each inputted obstacle/point the function from subtask 4. After the end of input output the list by calling the function from subtask 5. Last delete the whole list by calling the function from subtask 6.

Important to Regard : Only use C++ input and output, no calls of scanf or printf function, only new and delete (not malloc, calloc, realloc or free. Your file has to have name h2_yourMatriculationNumber.cpp, the ending .cpp is essential for the plagiarism checker as well as for our check programs, no text file with ending .txt, no project file .cbp or similar and no .rar, .zip, ... file (the plagiarism checker does not understand all these formats, therefore you would get 0 points for it), also your program is only allowed containing ASCII characters (no characters or signs from non-Latin ones), only includes of standard C++ libraries (i.e. no libraries with ending .h like conio.h, stdio.h, windows.h, ...) and shall follow standard C++11 (or newer). Start your file with a comment /* ... */ giving your personal data, not with / * ... * / or ... or completely without C++ comment tags.

Example 1 Program Run:

```
string describing obstacle ("end" for end of input): A
x and y coordinate: 0 1
string describing obstacle ("end" for end of input): X
x and y coordinate: 1 1
```

```
string describing obstacle ("end" for end of input): E
x and y coordinate: 0 3
string describing obstacle ("end" for end of input): K
x and y coordinate: -1 4
string describing obstacle ("end" for end of input): W
x and y coordinate: 0 10
string describing obstacle ("end" for end of input): end
obstacle A: (  0.00,   1.00), distance:  1.00m, nearest to this: X
obstacle X: (  1.00,   1.00), distance:  1.41m, nearest to this: A
obstacle E: (  0.00,   3.00), distance:  3.00m, nearest to this: K
obstacle K: ( -1.00,   4.00), distance:  4.12m, nearest to this: E
obstacle W: (  0.00,  10.00), distance: 10.00m, nearest to this: K
delete: A X E K W
```

Example 2 Program Run:

```
string describing obstacle ("end" for end of input): A
x and y coordinate: -27.2 12.8
string describing obstacle ("end" for end of input): B
x and y coordinate: 0.1 41.4
string describing obstacle ("end" for end of input): C
x and y coordinate: 15.9 30.25
string describing obstacle ("end" for end of input): D
x and y coordinate: -12.74 29.13
string describing obstacle ("end" for end of input): E
x and y coordinate: 27.68 23.45
string describing obstacle ("end" for end of input): F
x and y coordinate: 29.41 37.92
string describing obstacle ("end" for end of input): G
x and y coordinate: -21.03 45.19
string describing obstacle ("end" for end of input): H
x and y coordinate: 13.47 42.1
string describing obstacle ("end" for end of input): end
obstacle A: (-27.20,  12.80), distance: 30.06m, nearest to this: D
obstacle D: (-12.74,  29.13), distance: 31.79m, nearest to this: B
obstacle C: ( 15.90,  30.25), distance: 34.17m, nearest to this: H
obstacle E: ( 27.68,  23.45), distance: 36.28m, nearest to this: C
obstacle B: (  0.10,  41.40), distance: 41.40m, nearest to this: H
obstacle H: ( 13.47,  42.10), distance: 44.20m, nearest to this: C
obstacle F: ( 29.41,  37.92), distance: 47.99m, nearest to this: E
obstacle G: (-21.03,  45.19), distance: 49.84m, nearest to this: D
delete: A D C E B H F G
```