Note: For every exercise, create an own ~~project~~. At least one of the exercises has to be completed.

Learning outcome of this set of exercises: Develop and apply algorithms to solve problems, practice the implementation of functions and structures, ~~working with pointers and type casts~~.

Exercise categories:

- A – very basic, intended for inexperienced developers
- B – fair, a little bit more complex but still for starters
- C – challenging, complexity is higher, additional programming constructs may be required

Exercise 1 – Student database (A, B)

Write a program which manages a database containing students (A). The program allows to:

- add any number of students to the database
- print a list of student data (all data, selected data)
- search for students by matriculation number or name or by both
- sort the database by name or matriculation number

Use the data structure *student* (~~lesson day 4~~) and modify the data structure for *dynamic integer arrays* to define a data structure for *dynamic student arrays*, which can be used as student database. Write the basic functions `init_array()`, `put_val()`, `print_array()` and `free_array()` of the student database on the basis of the code of the corresponding dynamic integer array functions.
Hints: Add the files ~~*student.h* and *student.c*~~ to your project. Add three new files ~~*dynStudArray.h*, *dynStudArray.c* and *main.c*~~ to your project, copy the code of the structure and the functions of the dynamic integer array to the corresponding files and modify it. Whenever possible and suitable, call the functions for students in your implementation (e.g. `print_student()`).

Test the functions in the main program:

- declare a student database and eight students,
- ~~initialize the students database with a size of 5 and a growth of 5,~~
- add the students to the database,
- print the content of the database,
- free the database.

Example output:

```
array@0x22fc88: (8/10) +5 not sorted
Müller, Max                         123456
Münster, Andreas                    234567
Müller, Karin                       125656
Adams, Douglas                      424242
Müller, Max                         987654
Zuse, Konrad                        657892
Holmes, Sherlock                    999999
Curie, Marie                        897654
```

Add the following functions for the dynamic student array (B):

`find_student()`: The function takes a pointer to a student database, a name and/or a matriculation number. Name or matriculation number might be passed as 0. The function searches for students, which meet the criterion(s). If one or more students were found, it creates a new

dynamic student array, adds the students and returns the pointer to the new array. If nothing was found, it returns `NULL`. ~~(Note: The caller of the function is responsible to free the returned array.)~~

`sort_array()`: The function uses the *bubble sort algorithm* to sort a dynamic student array in ascending order. The caller can choose to sort the array by *matriculation number* or *name* (use an enumeration type). Depending on your experience, you may implement the function swapping the *content* of the array elements (more similar to the example code from the lesson, but possibly slower) or by swapping *pointers* to the array elements (a little bit more challenging, but possibly faster).

Hints:

To swap the content in the student array, it is useful to implement a `copy_student()` function, which copies the data of one structure to another.

At first, implement to sort by *matriculation number* (this is nearly the same implementation we used for the dynamic integer array).

To sort the array by *name*, implement a helper function which takes two strings and returns e.g.:
-1 if the left string has to be sorted before the right string (left is lower than right),
0 if both strings are equal,
1 if the left string shall be sorted after the right one.

Test the functions to get an output similar to this example:

```
Search for Müller, Max
array@0x3e27e8: (2/2) +2 not sorted
Müller, Max                     123456
Müller, Max                     987654

Search for 424242
array@0x3e27e8: (1/2) +2 not sorted
Adams, Douglas                  424242

Search for Müller, Max,123456
array@0x3e27e8: (1/2) +2 not sorted
Müller, Max                     123456

Search for Karl, Otto

sort by matriculation number
array@0x22fc88: (8/10) +5 sorted
Müller, Max                     123456
Müller, Karin                   125656
Münster, Andreas                234567
Adams, Douglas                  424242
Zuse, Konrad                    657892
Curie, Marie                    897654
Müller, Max                     987654
Holmes, Sherlock                999999

sort by name
array@0x22fc88: (8/10) +5 sorted
Adams, Douglas                  424242
Curie, Marie                    897654
Holmes, Sherlock                999999
Müller, Karin                   125656
Müller, Max                     123456
Müller, Max                     987654
Münster, Andreas                234567
Zuse, Konrad                    657892
```

Exercise 2 – Histogram (C)

The histogram and the dynamic integer array have to be programmed in a more general way.

The data structure and the functions of the dynamic integer array have to be modified, allowing the user to choose the type of data which he wants to save in the array. The type of the member `elem` has to be changed to `void*` and a new member `type` has to be added to the data structure of the dynamic array for saving the type of the data. Use an enumeration type and implement the types *integer* and *double* at first. Implement the following functions on the basis of the functions of the dynamic integer array:

- **`init_array()`**: You have to pass the type as an additional parameter.
- **`put_val()`**: To pass the value, work with void*: To store the value, cast the pointer according to the type of the array.
- **`print_array()`**
- **`sort_array()`**
- **`free_array()`**
- **`get_element()`**: Returns the value which is stored at a certain position. The value is returned as *double* value, regardless of the type of the array. If the position is invalid, it displays a warning and returns 0.
- **`get_min()`**, **`get_max()`**: Returns the *double* value of the smallest or largest element of the array.

Test the functions for both array types - fill it with random numbers, print it, sort it, print it once more, print the minimum and the maximum number of the arrays.
Hint: To get numbers of type *double* from `rand()`, you can use the constant `RAND_MAX`, which defines the maximum output of the random generator function.

Example output:

```
double test
-------------------------------
array@0x22ff14: (21/25) +5 DOUBLE not sorted
-0.375179 -0.479797  0.313318  0.063952  0.218711 -0.354671 -0.139576 -0.272332
 0.095569  0.061022 -0.476714 -0.137379 -0.240532 -0.455382 -0.398556  0.176229
-0.229728 -0.129688 -0.088702  0.301660  0.480895
array@0x22ff14: (21/25) +5 DOUBLE sorted
-0.479797 -0.476714 -0.455382 -0.398556 -0.375179 -0.354671 -0.272332 -0.240532
-0.229728 -0.139576 -0.137379 -0.129688 -0.088702  0.061022  0.063952  0.095569
 0.176229  0.218711  0.301660  0.313318  0.480895
Minimum:  -0.479797
Maximum:   0.480895

integer test
-------------------------------
array@0x22fefc: (21/25) +5 INT not sorted
      5       4       1       1       3       2       4       1       6       6
      4       3       1       2       2       6       3       1       3       1
      2
array@0x22fefc: (21/25) +5 INT sorted
      1       1       1       1       1       1       2       2       2       2
      3       3       3       3       4       4       4       5       6       6
      6
Minimum:          1
Maximum:          6
```

Use the new array type to program a more general histogram. The user shall be able to determine the number of bins. When the histogram is calculated for a certain array of data, the range of the data is calculated and subdivided into the specified number of bins. After this is done, the absolute frequency of the data is counted for each bin.

Define a data structure for histograms, consisting of a dynamic array of integer type for the absolute frequencies, a dynamic array of type double for the boundary values of the bins and a member variable for the current number of values the histogram is calculated from.

Write the following functions:

- **`init_histogram()`**: Takes the address of a histogram structure variable and the number of bins. It initializes the dynamic arrays of the structure according to the required number of bins and sets the current number of values to 0.
- **`calc_histogram()`**: Takes a histogram structure and a dynamic array of any type containing the values the histogram has to be calculated from (both parameters passed by pointer). It calculates the boundaries of the bins and the absolute frequency for each bin and sanes it in the corresponding member of the histogram structure. It also saves the number of values the histogram was calculated from.
  Hint: The bins are stored starting with the minimum and ending with the maximum value of the passed array. The absolute frequency of the first bin is the frequency of the values which are greater or equal the minimum value and lower the boundary stored in position 1 of the bins array, the absolute frequency of the second bin is the frequency of the values which are greater or equal the boundary stored in position 1 and lower than the boundary stored in position 2, and so on.
- **`print_histogram()`**: Displays the number of bins and the minimum and maximum value of the data the histogram was calculated from. Displays the absolute or relative frequencies (the user may choose) with the corresponding boundaries and centers of the bins (see example).
- **`free_histogram()`**: frees the dynamic arrays for the frequencies and bins.

Test the histogram functions by calculating the histogram of the arrays from the preceding test. Print the histograms with relative and absolute frequencies.

Example output:

```
histogram test
------------------------------
array@0x22ff14: (21/25) +5 DOUBLE sorted
-0.488250 -0.439940 -0.428556 -0.328333 -0.219413 -0.211264 -0.203024 -0.201102
-0.069353 -0.031205 -0.029344  0.012864  0.021897  0.030229  0.054674  0.077654
 0.172323  0.469848  0.481811  0.490387  0.496979

Histogram (number of bins/minimum of data/maximum of data): 5/-0.488250/0.496979
boundary bin | center bin | abs. frequency
-------------+------------+---------------
   -0.488250 |      -0.39 |             4
   -0.291205 |      -0.19 |             4
   -0.094159 |       0.00 |             8
    0.102887 |       0.20 |             1
    0.299933 |       0.40 |             4
    0.496979 |            |

Histogram (number of bins/minimum of data/maximum of data): 5/-0.488250/0.496979
boundary bin | center bin | rel. frequency
-------------+------------+---------------
   -0.488250 |      -0.39 |   0.190476
   -0.291205 |      -0.19 |   0.190476
   -0.094159 |       0.00 |   0.380952
    0.102887 |       0.20 |   0.047619
    0.299933 |       0.40 |   0.190476
    0.496979 |            |
```