CS 392: Homework Assignment 2
Due: February 19, 11:55pm

Philippos Mordohai
Department of Computer Science
Stevens Institute of Technology
Philippos.Mordohai@stevens.edu

Collaboration Policy. Homeworks will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems. Use of the Internet is allowed, but should not include searching for previous solutions or answers to the specific questions of the assignment.

Late Policy. No late submissions will be allowed without consent from the instructor. If urgent or unusual circumstances prevent you from submitting a homework assignment in time, please e-mail me explaining the situation.

1 Preliminaries

Concepts that are relevant to this assignment:

- C project file dependencies and makefile;
- The bubble sort algorithm;
- Dynamic memory allocation and memory leaks;
- Reading files using `FILE*`;
- Void pointers and function pointers.

Your task is to write an bubble sort algorithm that can sort an array of generic data, given appropriate helper functions.

2 File Structure

In the starter code, you are provided with the following five files:

- `main.c`: for testing your code;
- `bubble.h`: declares data types and functions needed for bubble sort;

- `bubble.c`: implements the bubble sort algorithm;
- `utils.h`: declares type-specific functions;
- `utils.c`: implements type-specific functions;
- `Makefile`: the makefile for compiling your code. (not provided)

The red colored files are the ones you need to complete.

3 Utility Functions (`utils.h`)

A sorting algorithm constantly compares data to decide the relative order of any two elements in the array. In this assignment, the bubble sort algorithm needs to be generic and able to work with any data type. Therefore, when calling the bubble sort function declared in `bubble.h`, we have to pass pointers to type-specific functions to it. In this assignment, you will need to create type-specific functions for two types: `int` and `double`:

The first function is for comparing two data elements. It returns the sign of their difference, i.e. -1, 0 or +1.

```
int cmpr_int(void*, void*);
int cmpr_double(void*,void*);
```

`cmpr_int()` compares two integers, while `cmpr_double()` compares two double precision numbers.

The following print functions print **one** element of the array by calling `printf()` with the appropriate format.

```
void print_int(void*);
void print_double(void*);
```

Lastly, you will create a function to read all the elements in an array from a file:

```
void* read_array(char* filename, char* format, size_t* len);
```

where `filename` is the name of the file where the array is stored. You can safely assume all the numbers in the file are valid, and each line contains one number. The filename has to be passed as command-line argument `argv[1]`. If the file cannot be successfully opened, print the following error message through `stderr` and exit with code 1:

```
File failed to open.
```

The argument `format` is passed from `argv[2]`, and can be either `"%d"` or `"%lf"` for integers and doubles, respectively. The argument `len` stores the length of the array. (It is an output of the function.)

The function returns a pointer pointing to the array created in it.

In summary, the five functions you need to complete are:

- `cmpr_int();`
- `cmpr_double();`
- `print_int();`
- `print_double();`
- `read_array().`

4 Bubble Sort (`bubble.h`)

There are two functions you need to implement for bubble sort.

4.1 Main Sorting Algorithm: `bubble_sort()`

The function declaration is as follows:

```
void bubble_sort(void* base, size_t nel, size_t width, int (*compare)(void*,void*));
```

The arguments are used this way:

1. `void* base`: this is where you pass the base address of the array;
2. `size_t nel`: indicates the number of elements in the array;
3. `size_t width`: the size of each element;
4. `int (*compare)(void*,void*)`: a function pointer pointing to a type-specific `cmpr` function in `utils.h`.

4.2 Printing the Array: `bubble_print()`

The function `bubble_print()` prints all the elements in an array, and is declared as follows:

```
void bubble_print(void* base, size_t nel, size_t width, void (*print)(void*));
```

where the first arguments are the same as in `bubble_sort()`. Based on the data type of the array, you should pass a type-specific `print` function declared in `utils.h`.

5 Testing (`main.c`)

`main.c` provides a simple test case for integer data that can be found in the `testint` file. The output should be:

```
1
3
5
10
15
20
```

You will also need to test the `double` case on your own. You can modify `main.c` as you wish, but for grading your code will be tested with a separate, different `main.c`. To ensure that your code will work with the grading script, please follow the example provided in `main.c` closely.

6 Makefile

You should also write a Makefile to compile your code. Make sure your main function is in `main.c`.

For initial testing you can compile your program from the terminal using:

```
gcc main.c bubble.c utils.c -o bubblesort
```

7 Read This Before You Start (Requirements)

- `bubble.c`
  Very important: because this is a generic implementation, in `bubble.c` you **must not** use `if-else` to discriminate different data types in your code. Your code must be able to deal with any data type that supports subtraction. Therefore, we limit the usage of the following primitive types:

  - `int` or `size_t`: only allowed for loop iterators;
  - `char`: can be used as you wish;
  - Types that are used in the declarations of `bubble_sort()` and `bubble_print()`: feel free to use them for their *original* purpose;
  - All other types are not allowed in `bubble.c`: **no exceptions**.

  You are free to create helper functions **only** in `bubble.c`; you must declare and implement them in the `.c` files instead of the header files. Do not change any of the header files.

- `utils.c`
  You are free to use any data type you like in this file. However, other than the five functions declared in `utils.h`, do not create any other functions. Do not change any of the header files.

- Other requirements:
  - That does not compile, hangs, crashes due to segmentation or other faults, etc. will receive no credit;
  - No memory leaks. The array to be sorted will be freed by us in the testing script, so you do not need to worry about that. However, there must be no other memory leaks, and when we free the pointer returned by your `read_array()` function, there must be no errors;
  - You do not need to include any other header files. However, if you do, make sure you can compile it successfully from your Makefile;
  - When in doubt (*e.g.,* "Can we assume...?"), please do not hesitate to ask. Pay attention to Canvas announcement closely.

8 Deliverables

Zip the following files and submit on Canvas:

1. `bubble.c`;
2. `utils.c`;
3. `Makefile`.